**POLITECHNIKA WARSZAWSKA**

**Wydział Elektroniki i Technik Informacyjnych**

**Instytut Informatyki**

**mgr inż. Przemysław Sołdacki**

# ROZPRAWA DOKTORSKA

# Zastosowanie metod płytkiej analizy tekstu do przetwarzania dokumentów w języku polskim

**Promotor:**

**Prof. dr hab. inż. Mieczysław Muraszkiewicz**

**Warszawa 2006**

## Streszczenie

Niniejsza rozprawa poświęcona jest tematyce automatycznego przetwarzania dokumentów w języku naturalnym, a w szczególności zastosowania metod płytkiej analizy tekstów do przetwarzania dokumentów w języku polskim. Jest to dziedzina z pogranicza przetwarzania języka naturalnego i eksploracji danych.

Podstawowym celem tej rozprawy było zaproponowanie nowych metod analizy dokumentów polskich w oparciu o płytką analizę tekstu. W tym celu opracowano i zaimplementowano algorytmy i odpowiednie narzędzia oraz przeprowadzono próbę poprawy wyników wybranych zadań eksploracji tekstu.

Na początku rozprawy przedstawiono wprowadzenie do jej tematyki, podano używane pojęcia oraz omówiono rozwiązania proponowane w literaturze. Następnie, przedstawiono model płytkiej analizy tekstu oraz omówiono najczęściej spotykane zagadnienia eksploracji tekstu, po czym bardziej szczegółowo omówiono metody reprezentacji tekstu, klasyfikację, grupowanie oraz sumaryzację. W dalszej części rozprawy przedstawiono szczegółowo etapy płytkiej analizy tekstu wraz z propozycjami rozwiązań dla języka polskiego. W ramach przygotowanej platformy badawczej przeprowadzono również implementację i testy zaproponowanych rozwiązań. Uzyskane wyniki są obiecujące, pozostawiają także duże pole do dalszych badań.

Przeprowadzono również próbę wykorzystania wyników płytkiej analizy tekstu do poprawy wyników klasyfikacji, grupowania i sumaryzacji. Uzyskano poprawę wyników.



**Słowa kluczowe**: płytka analiza tekstu, eksploracja tekstu, klasyfikacja, grupowanie, sumaryzacja, text mining, NLP.

# Spis treści

# Od autora

W obecnych czasach komputery stały się stałym elementem naszego świata. Trudno również sobie wyobrazić sprawne funkcjonowanie we współczesnym świecie bez dostępu do Internetu, jego zasobów informacyjnych i możliwości komunikacyjnych. Coraz więcej spraw może a nawet musi być załatwianych przez Internet. Komunikując się ludzie wymieniają coraz większe ilości informacji. Aby sobie z tym poradzić potrzebują coraz lepszych narzędzi, które będą ich wspomagać w tym zakresie. Powstają coraz bardziej wyrafinowane narzędzia do wyszukiwania i filtrowania informacji. Ze względu na międzynarodowy charakter Internetu ważne jest uwzględnianie specyfiki występujących w nim języków. Powyższe fakty stanowią podstawę mojego zainteresowania tematyką automatycznego przetwarzania języka polskiego. Uzyskane wyniki przeprowadzonych eksperymentów, o których piszę w rozprawie są bardzo obiecujące, co pozwala mi wyrazić przekonanie, że niniejsza praca przyczyni się do rozwoju tej dziedziny oraz do jej spopularyzowania.

**Podziękowania**

Chciałbym podziękować prof. dr hab. inż. Mieczysławowi Muraszkiewiczowi, bez którego wsparcia i pomocy ta rozprawa by nie powstała. Wyrazy wdzięczności składam również wszystkim uczestnikom prowadzonych przeze mnie projektów badawczych w ramach przedmiotu „Inteligentne Systemy Informacyjne” prowadzonego przez prof. M. Muraszkiewicza na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej.

# 1. Wprowadzenie

## *1.1. Motywacja*

Rozwój cywilizacji ludzkiej w dużym stopniu opiera się na zdolności do gromadzenia, przekazywania i powtórnego wykorzystania informacji. Początkowo informacje były przekazywane w postaci ustnej. Wynalazek pisma pozwolił na gromadzenie coraz większych ilości informacji. Kolejnym krokiem rozwoju było wynalezienie przez Guttenberga druku, który pozwolił na szerokie rozprzestrzenianie informacji. Współcześnie wykorzystanie mediów elektronicznych pozwala na coraz tańszy, a przez to łatwiejszy dostęp do informacji. Są one gromadzone i udostępniane przez coraz szersze grona użytkowników. Udostępnianie informacji staje się w pewnym stopniu zachowaniem społecznym [Muraszkiewicz, 2002] Oprócz informacji tworzonych przez ludzi coraz częściej mamy do czynienia z informacjami tworzonymi automatycznie, np. z danych pomiarowych, systemów transakcyjnych itp. Biorąc pod uwagę tempo przyrostu ilości dostępnych informacji można mówić o tzw. **eksplozji informacji.**

Obszerne zasoby dokumentów tekstowych są przechowywane w systemach zarządzania kontaktami z klientem (ang. ***Customer Relationship Management - CRM***), serwerach pocztowych, portalach informacyjnych itp. Duże ilości nieustrukturalizowanych danych takich jak oferty, umowy, raporty, polisy ubezpieczeniowe, regulaminy są gromadzone na wielu serwerach firm i organizacji.

Internet z wielkimi zasobami informacyjnymi jest potencjalnie najszybszym źródłem pozyskiwania informacji. Współczesne firmy i organizacje stają przed zadaniem monitorowania tych olbrzymich zasobów. Jednocześnie pośród tego zalewu informacji coraz trudniejsze jest oddzielenie informacji ważnych od nieistotnych. Jeszcze trudniejszym zadaniem jest **pozyskiwanie wiedzy**, czyli informacji możliwej do bezpośredniego wykorzystania w danej sytuacji. Wiedza staje się kluczowym elementem w większości dziedzin współczesnego życia. Jest ona doceniana zarówno przez środowiska naukowe, jak i biznesowe. Wobec tego bardzo istotny jest rozwój technik pozwalających wspierać ludzi w procesie uzyskiwania wiedzy z dużych zasobów informacyjnych.

Większość zasobów Internetu ma postać dokumentów tekstowych (HTML) bez określonej struktury, przez co ich automatyczne wyszukiwanie lub przetwarzanie jest utrudnione. Z tego powodu rozwijane są zarówno techniki ułatwiające wyszukiwanie i prezentację informacji operatorowi, jak również techniki pozwalające na strukturalizację informacji w celu ich dalszego automatycznego przetwarzania.

Pewną pomocą w przeszukiwaniu tego gigantycznego repozytorium są indeksy (np. Yahoo!), przewodniki (np. About.com) czy wyszukiwarki internetowe typu Google. Jednak ze względu na ciągły wykładniczy wzrost ilości informacji przechowywanych na serwerach internetowych konieczne jest tworzenie coraz szybszych, bardziej precyzyjnych i inteligentnych systemów pozwalających na wyszukiwanie, klasyfikowanie i streszczanie informacji.

## *1.2. Cel*

Istotną cechą większości algorytmów i rozwiązań do eksploracji tekstów jest ich zależność od języka. Systemy stworzone dla języka angielskiego są często mało przydatne do przetwarzania dokumentów polskich. Większość narzędzi typu *text mining* obsługuje pozostałe języki jedynie na poziomie rozpoznawania liter z różnych stron kodowych. Niektóre narzędzia potrafią przeprowadzać pewne transformacje lingwistyczne, np. sprowadzanie wyrazów do formy podstawowej (ang. ***stemming***), jednak większość z nich nie obsługuje języka polskiego (*Statistica Text Miner*, *Google*, *IBM Intelligent Miner for Text*, *SemioMaps*, narzędzia do tłumaczeń i analizy leksykalnej firm *Teragram Corp.* i *Systran*). O ile w niektórych systemach implementowane są pewne mechanizmy wielojęzyczności (np. *Google*), to jednak wciąż brakuje skutecznych narzędzi obsługujących całą specyfikę języka polskiego, jak chociażby fleksję.

Obecnie większość publikacji i artykułów naukowych dotyczy przetwarzania języka angielskiego. Coraz częściej pojawiają się również prace na temat innych języków indoeuropejskich, a także azjatyckich. W ostatnim czasie również automatyczne przetwarzanie dokumentów w języku polskim jest tematem intensywnych badań w kilku ośrodkach w Polsce. Tematyka automatycznego przetwarzania języka polskiego wciąż wymaga dalszych badań. Wciąż brakuje literatury zarówno napisanej w języku

polskim, jak również poruszającej problem przetwarzania języka polskiego w kontekście szeroko pojętej eksploracji tekstu.

Jednym z celów niniejszej rozprawy jest przedstawienie aktualnego stanu wiedzy na temat omawianej dziedziny, przez co zwiększenia jej dostępności dla polskich czytelników. Równie istotnym aspektem jest przedstawienie technik przetwarzania języka polskiego. Omawiane zagadnienia mogą być w przyszłości wykorzystane w wielu technikach związanych z automatycznym przetwarzaniem dokumentów polskich. Również narzędzia programowe, które powstały przy okazji tworzenia niniejszej rozprawy mogą okazać się przydatne.

Pisząc pracę w języku polskim należy w możliwie dużym stopniu wykorzystywać polską terminologię. Nie jest to zadanie proste w przypadku, gdy większość określeń na temat tej dziedziny funkcjonuje w postaci anglojęzycznej. W miarę możliwości zastosowani polskie odpowiedniki, jednak dla zachowania pełnej zgodności z istniejącą literaturą podano również terminy angielskie. W szczególności użyto skróty pochodzące od terminów angielskich. Pozwala to na sprawniejsze posługiwanie się swoistym żargonem używanym przez specjalistów z omawianej dziedziny.

Rozprawa ta w sposób bardziej szczegółowy porusza wybrany element dziedziny, jakim jest płytka analiza tekstu (ang. ***shallow text processing***) oraz jej zastosowanie do rozwiązywania zadań z zakresu eksploracji tekstów. Celem jest stworzenie platformy badawczej składającej się z algorytmów oraz zestawu narzędzi programowych automatycznego przetwarzania języka polskiego, w szczególności opartych na metodach płytkiej analizy tekstu. Obejmuje to także opracowanie odpowiedniego środowiska narzędziowego i odpowiednich interfejsów, jak również przygotowanie narzędzi pracujących w tym środowisku, co niechybnie ułatwi tworzenie kolejnych narzędzi.

## *1.3. Text Mining – eksploracja tekstu*

Text Mining (określany również jako Text Data Mining - ***TDM***) jest to dziedzina zajmująca się przetwarzaniem zbiorów dokumentów w celu znalezienia informacji, która nie jest dostępna bezpośrednio lub też jest trudno dostępna. Jest to sposób

znajdywania nowej wiedzy pośród olbrzymich zasobów tekstowych, które stały się dostępne dla przetwarzania komputerowego.

Techniki wykorzystywane w TDM są pod pewnymi względami podobne do technik znanych ze standardowej eksploracji danych (ang. ***Data Mining***) pod tym względem, że służą do przetwarzania bardzo dużych ilości danych. Obie grupy technik pozwalają na odkrywanie istotnych zależności oraz informacji występujących w zbiorach danych, które nie są dostępne bezpośrednio, a także służą pomocą w ich zrozumieniu.

Główna różnica między technikami z grupy Data Mining i Text Mining polega na tym, że pierwsza z nich analizuje, przekształca i podsumowuje głównie dane numeryczne tworząc ich modele matematyczne. Natomiast druga przetwarza duże zasoby nieustrukturalizowanych danych tekstowych.

Do typowych zadań Data Mining'u zalicza się klasyfikację i grupowanie, wizualizację, a także wykrywanie zbiorów częstych i reguł asocjacyjnych. Opracowano bardzo wydajne algorytmy realizacji tych zadań (w szczególności prace Agrawal'a), m.in. wykrywanie reguł asocjacyjnych w oparciu o zwięzłe reprezentacje [Kryszkiewicz, 2000], [Kryszkiewicz, 2001] czy techniki oparte o bazy danych [Królikowski, Morzy, Perek, 2003] ,[Morzy, Morzy, Królikowski, 2004].

Text Mining wciąż znajduje się w początkowej fazie rozwoju, mimo to dotychczasowe osiągnięcia są bardzo obiecujące. W bliskiej przyszłości ma on szansę stać się jedną z kluczowych technologii dla wielu gałęzi biznesu, ponieważ bardzo często firmy i przedsiębiorstwa „nie wiedzą czego nie wiedzą" [Tkach, 1999]. Dalsze badania potwierdzają przydatność technik Text Mining'u w zastosowaniach praktycznych [Gawrysiak, Rybiński, Okoniewski, 2004], [Gawrysiak, Rybiński, Gajda, 2004], [Kierski, Okoniewski, Gawrysiak, 2003]. Dostępnych jest również coraz więcej narzędzi komercyjnych korzystających z różnych technik Text Mining, np. wyszukiwarki internetowe, narzędzia Google, MSSearch czy narzędzia klasyfikacji tekstu w serwerze MS SharePoint Portal Server.

Rozszerzeniem technik TDM jest tzw. ***web mining***, który skupia się na analizie stron internetowych oraz logów serwerów WWW w celu znalezienia wzorców zachowań użytkowników przeglądających strony internetowe. Takie informacje są często wiązane z dokumentami tekstowymi, wiadomościami e-mail, arkuszami kalkulacyjnymi czy

bazami korporacyjnymi w celu tworzenia portali intranetowych. Tego typu techniki pozwalające na łączenie wiedzy z procesami jej przetwarzania i wykorzystywania określa się jako **Zarządzanie Wiedzą** (ang. ***Knowledge Management*** - ***KM***). Chociaż systemy KM powstawały w celu propagowania informacji wewnątrz korporacji, to z czasem zaczęły ulegać rozszerzaniu. W ten sposób zaczęły powstawać centra informacji naukowej oraz przemysłowej czy wirtualne biblioteki. Przechowują one różnorodne informacje zarówno w postaci ścisłych struktur danych, jak i swobodnych dokumentów tekstowych. Stanowi to kolejne wyzwanie dla systemów wyszukiwania informacji.

### 1.3.1. Zadania TDM

Poniżej znajduje się spis najczęściej występujących zadań TDM.

**Klasyfikacja dokumentów** (ang. ***Document classification***) zwana również **kategoryzacją dokumentów** (ang. ***Document categorization***) polega na przypisywaniu nowych dokumentów do jednej lub kilku z predefiniowanych klas. Takie klasy mogą zostać opisane ręcznie przez ekspertów lub też mogą być zdefiniowane poprzez zbiory dokumentów już do nich należących. W większości przypadków klasy nie są zagnieżdżane, natomiast przyjmuje się, iż jeden dokument może należeć do więcej niż jednej klasy [Yang, 1998]. Klasyfikacja jest bardzo pomocna podczas wyszukiwania oraz filtrowania informacji.

Problemem pokrewnym do klasyfikacji jest grupowanie dokumentów. W tym przypadku system nie posiada wejściowej wiedzy, w postaci już zaklasyfikowanych dokumentów, czy też samych klas. W tym przypadku grupy nie są dane i są one wynikiem realizacji zadania. **Grupowanie dokumentów** (ang. ***Document clustering***) polega na próbie połączenia w grupy (klastry) dokumentów na podstawie ich podobieństwa, tak aby dokumenty dotyczące jednego tematu trafiły do tej samej grupy. Innymi słowy zadanie polega na takim pogrupowaniu dokumentów, by dokumenty należące do jednej klasy były do siebie możliwie podobne i jednocześnie różniły się znacząco od dokumentów należących do innych klas [Sholom, 2000]. Grupowanie to może być hierarchiczne, wtedy poszczególne grupy mogą być zagnieżdżane. Tworzone w procesie grupowania klasy mogą być później wykorzystywane do klasyfikowania nowych dokumentów. Takie pogrupowanie jest pomocne podczas szukania informacji,

a także pozwala na szybką orientację w zakresie i strukturze tematycznej dużego zbioru dokumentów.

Automatyczne grupowanie i klasyfikacja dokumentów tekstowych wywodzą się z technik eksploracji danych, czyli grupowania i klasyfikowania rekordów z baz danych. Obie te metody okazują się być bardzo przydatne w analizie bardzo dużych zbiorów danych. W przypadku obu zadań mamy do czynienia z procesem uczenia maszynowego. Różnica polega na tym, iż w pierwszym przypadku jest to uczenie z nadzorem, a w drugim – bez nadzoru.

**Sumaryzacja** (ang. ***Summarization***) polega na generowaniu streszczeń. W zależności od sposobu postawienia zadania streszczenie może obejmować treść jednego dokumentu lub zbioru dokumentów. Podstawowym założeniem jest uzyskanie streszczenia, które zawiera najważniejsze informacje z tekstu źródłowego. Zwykle rozmiar uzyskiwanego streszczenia jest jednym z parametrów zadania. Rozwiązanie tego problemu jest bardzo przydatne do analizy dużych zbiorów tekstowych. Zwalniając operatora z czytania i analizy całości tekstu, uzyskuje się możliwość skupienia jego uwagi na aspektach, których rozwiązanie w sposób automatyczny jest niemożliwe lub nieskuteczne.

**Automatyczne rozpoznawanie języka** (ang. ***Automatic Language Identification***): Ta grupa technik pozwalających na rozpoznanie języka, w jakim został napisany dokument, odgrywa bardzo dużą rolę w systemach wielojęzycznych. Umożliwia to na efektywne wykorzystanie innych technik TDM w środowiskach wielonarodowych i wielokulturowych. Prostym przykładem wykorzystania takich technik jest rozpoznawanie języka zdań w edytorze tekstu w procesie sprawdzania poprawności pisowni.

**Grupowanie pojęć** (ang. ***Concept clustering***): Algorytmy należące do tej grupy wykrywają zależności pomiędzy słowami występującymi w zbiorze dokumentów w celu znalezienia synonimów, budowy tezaurusów i słowników, wskazania słów kluczowych, automatycznego rozpoznawanie autora po stylu pisania itp.

**Wizualizacja i nawigacja**: Graficzna prezentacja treści dokumentu lub też zbioru dokumentów może znacznie ułatwić poruszanie się w dużym zbiorze dokumentów. Pozwala to również na skuteczniejszą prezentację wyników wyszukiwania.

***Web Mining***: Techniki TDM mogą przetwarzać różnorodne dokumenty tekstowe, w tym także strony internetowe dostępne on-line. Jednak strony WWW oprócz tekstu zawierają znacznie więcej potencjalnie przydatnych informacji. W szczególności interesujące są wzajemne odwołania stron, które mogą świadczyć o zależności tematycznej oraz pozwalają oszacować ważność stron. Przykładowo strony, do których jest dużo odwołań, mogą być bardziej interesujące. Z tego powodu wyodrębnia się grupę technik służących do ekstrakcji wiedzy ze stron WWW, określaną jako web mining.

**Wyszukiwanie informacji** (ang. ***Information Retrieval, IR***): Zadanie to polega na znalezieniu w dużym zbiorze dokumentów tekstowych podzbioru dokumentów relewantnych na podstawie zapytania użytkownika. Klasycznym przykładem realizacji tego zadania jest wyszukiwarka WWW. Współczesne algorytmy, poprzez oszacowanie ważności dokumentów, a także znajdywanie grup silnie powiązanych dokumentów lub stron webowych, pozwalają na uzyskanie wysokiej skuteczności oraz wydajności znajdywania dokumentów. Dzięki temu możliwe jest tworzenie zaawansowanych narzędzi wyszukiwania i wizualizacji dokumentów, a także na automatyczne tworzenie indeksów.

Wyszukiwanie informacji (IR) pomimo swojej niewątpliwej przydatności w pewnych okolicznościach okazuje się niewystarczające. W wielu zastosowaniach oprócz znajdywania relewantnych dokumentów potrzebne jest odseparowanie konkretnych informacji i koncepcji. Z tego powodu rozwinęła się nowa dziedzina zwana ekstrakcją informacji.

**Ekstrakcja informacji** (ang. ***Information Extraction***, ***IE***) polega na identyfikacji predefiniowanego zbioru wyrażeń z określonej dziedziny z pominięciem innych nieistotnych informacji, przy czym dziedzina jest zdefiniowana poprzez korpus zbiorów tekstowych oraz wyraźnie określenie poszukiwanych informacji. Uzyskiwane informacje są predefiniowane w postaci wzorców zdefiniowanych przez użytkownika, np. informacja o firmie, o produkcie, spotkanie. Każdy wzorzec składa się ze zbioru pól (ang. ***slots***), które muszą zostać skonkretyzowane podczas przetwarzania tekstu. Pola są zwykle wypełniane przez ciąg znaków z tekstu, jedną z predefiniowanych wartości lub odnośnikiem do już wcześniej skonkretyzowanego wzorca. Ekstrakcja informacji może

być postrzegana jako przetwarzanie postaci informacji ze swobodnej tekstowej na sformalizowaną bazodanową, gdyż struktura wzorców jest z góry założona.

Nawet przy bardzo prostej dziedzinie zadanie to nie jest łatwe z powodu złożoności języka naturalnego. Przykładowo taka sama informacja może być wyrażona na wiele sposobów, a dodatkowo może być rozrzucona po kilku zdaniach. Próby rozwiązania tego zadania dla języka polskiego zostały podjęte m.in. w [Piskorski, Homola, 2004].

Duży wpływ na rozwój ekstrakcji informacji miały konferencje ***MUC*** (***Message Understanding Conferences***), które odbywały się pod auspicjami kilku agencji rządowych w USA i miały na celu zintegrowanie grup badawczych w zakresie IE oraz IR. Podczas tych konferencji zdefiniowano wiele zadań IE, które stały się punktem odniesienia do oceny działania systemów. Zdefiniowane zostały precyzyjne specyfikacje, formaty reprezentacji informacji oraz zbiór korpusów danych. Więcej informacji na ten temat można znaleźć w [Piskorski, 2001].

## 1.4. NLP – przetwarzanie języka naturalnego

Pojęciem ***NLP*** (ang. ***Natural Language Processing***) określa się zbiór technik komputerowych służących do analizy i reprezentacji tekstów występujących na poziomie analizy lingwistycznej w celu uzyskania sposobu przetwarzania języka przypominającego ludzki w określonym zakresie zadań i zastosowań.

Ponieważ TDM korzysta z metod statystycznych do analizy tekstu, to bywa czasami postrzegany jako rozszerzenie NLP. Występują jednak pewne różnice pomiędzy TDM a klasycznym NLP. Najważniejsze różnice przedstawia poniższa tabela.

| **TDM** | **NLP** |
|---|---|
| Przetwarzanie dużych zbiorów dokumentów tekstowych | Zwykle przetwarzanie pojedynczych dokumentów |
| Wykrywanie wcześniej nieznanych zależności i relacji występujących w tekstach | Przetwarzanie danych w oparciu o predefiniowane wzorce lub gramatyki |
| W większości automatyczne | W większości kontrolowane przez użytkownika |

## 1.5. Głęboka a płytka analiza tekstu

Potencjalna przydatność automatycznej analizy języka naturalnego została zauważona już w latach 50-tych XX wieku. Obecnie techniki NLP są wykorzystywane do wzbogacania rozwiązań TDM. Przykładowo reprezentacja tekstu uzupełniona informacjami leksykalnymi pozwala uzyskać skuteczniejszą klasyfikację lub grupowanie dokumentów. Niestety nadal wiele zagadnień NLP nie jest rozwiązywanych w sposób zadowalający. W szczególności dużym problemem, pojawiającym się na wielu etapach przetwarzania NLP, jest występująca w językach naturalnych niejednoznaczność. Słowa mogą mieć wiele znaczeń w zależności od szeroko pojmowanego kontekstu. Niejednoznaczności dotyczące granic wyrażeń i zdań

oraz ich wzajemnych relacji dodatkowo komplikują analizę. Przeprowadzenie pełnej analizy tekstu może być bardzo czasochłonne [Cole, Mariani, 1996]. Z tego powodu często stosowanym rozwiązaniem jest ograniczanie zakresu analizy.

**Głęboka analiza tekstu** (ang. ***Deep Text Processing***, ***DTP***) jest procesem komputerowej analizy lingwistycznej wszystkich możliwych interpretacji i relacji gramatycznych występujących w tekście naturalnym. Taka pełna analiza może być bardzo złożona. Co więcej w wielu wypadkach uzyskiwana w ten sposób informacja może nie być konieczna. Z tego powodu coraz częściej występuje tendencja do przeprowadzania jedynie częściowej analizy tekstu, która może być znacznie mniej czasochłonna i jest kompromisem między precyzją a wydajnością.

**Płytka analiza tekstu** (ang. ***Shallow Text Processing***, ***STP***) może być krótko określona jako analiza tekstu, której efekt jest niepełny w stosunku do głębokiej analizy tekstu [Piskorski, 2001]. Zwykle ograniczenie polega na rozpoznawaniu struktur nierekurencyjnych lub o ograniczonym poziomie rekurencji, które mogą być rozpoznane z dużym stopniem pewności. Struktury wymagające złożonej analizy wielu możliwych rozwiązań są pomijane lub analizowane częściowo. Analiza skierowana jest głównie na rozpoznawanie nazw własnych, wyrażeń rzeczownikowych, grup czasownikowych bez rozpoznawania ich wewnętrznej struktury i funkcji w zdaniu. Dodatkowo rozpoznawane są pewne główne części zdań, np. orzeczenie lub grupa orzeczenia.

Wykorzystywanie płytkiej analizy tekstu (zamiast głębokiej) może w wielu wypadkach okazać się wystarczająco do uzyskania potrzebnych informacji, a dzięki wprowadzeniu uproszczeń pozwala na uzyskanie oszczędności czasu. Poza tym przeprowadzenie głębokiej analizy prowadzącej do pełnego rozpoznania semantyki jest wciąż niemożliwe. Pomimo pewnych uproszczeń płytka analiza jest procesem złożonym, wymagającym rozwiązania wielu zagadnień lingwistycznych i stworzenia zestawu niezbędnych narzędzi.

## *1.6. Teza pracy*

Podczas prac nad niniejszą rozprawą sprawdzono możliwości przeprowadzenia płytkiej analizy tekstu w języku polskim oraz przeanalizowano wybrane aspekty automatycznego przetwarzania dokumentów tekstowych w języku polskim. Pozwala to na sformułowanie następującej tezy:

*Automatyczne przeprowadzenie kolejnych etapów płytkiej analizy tekstu w języku polskim jest możliwe i daje pozytywne efekty, a uzyskane w ten sposób informacje lingwistyczne mogą być wykorzystane do poprawy rezultatów w zadaniach eksploracji tekstów, takich jak klasyfikacja, grupowanie i sumaryzacja.*

W ramach prac badawczych wykonano m.in.:

- zaproponowano rozszerzony model płytkiej analizy tekstu w kontekście języka polskiego,
- opracowano algorytmy wykorzystywane na kolejnych etapach modelu,
- opracowano platformę badawczą definiującą moduły realizujące kolejne etapy przetwarzania oraz standardy komunikacji między nimi,
- stworzono narzędzia programowe działające w ramach opracowanej platformy oraz przetestowano ich skuteczność,
- zaproponowano własną metodę oceny, grupowania,
- sprawdzono możliwości wykorzystania dodatkowych informacji lingwistycznych do poprawy wyników zadań Text Mining'u na przykładzie klasyfikacji, grupowania i sumaryzacji.

## 1.7. Układ pracy

Struktura niniejszej rozprawy jest następująca:

W rozdziale pierwszym przedstawiono wprowadzenie do tematyki eksploracji tekstu. Omówiono różnice między technikami Text Mining'u oraz NLP oraz przedstawiono typowe zadania Text Mining'u. Wprowadzono również pojęcia głębokiej i płytkiej analizy tekstu.

W rozdziale drugim omówiono reprezentacje tekstu wykorzystywane w eksploracji tekstów oraz metody ich przekształcania. Przedstawiono również podstawowe algorytmy klasyfikacji i grupowania dokumentów. Omówiono również problem oceny jakości klasyfikacji i grupowania. Zaprezentowano także tematykę sumaryzacji i algorytmy wykorzystywane do realizacji tego zadania.

W rozdziale trzecim przedstawiono model płytkiej analizy tekstu wraz z zaproponowanymi rozszerzeniami i algorytmami. Zaprezentowano sposoby realizacji modelu dla języka polskiego. Przedstawiono testy praktycznych implementacji zaproponowanych rozwiązań.

W rozdziale czwartym opisano możliwości wykorzystania dodatkowych informacji lingwistycznych uzyskanych w procesie płytkiej analizy tekstu do realizacji wybranych zadań eksploracji tekstu. Przedstawiono wyniki testów na rzeczywistych danych.

W rozdziale piątym podsumowano uzyskane wyniki. Wyciągnięto wnioski z przeprowadzonych badań i wskazano kierunki dalszych badań.

Na końcu pracy znajdują się załączniki ze szczegółami implementacyjnymi, wynikami, a także spisy literatury, spis projektów badawczych i indeksy.

# 2. Techniki eksploracji tekstu

## *2.1. Reprezentacje*

Podstawowym problemem, jaki występuje w przypadku wielu zadań TDM (np. klasyfikacji i grupowania) jest wybór reprezentacji dokumentów tekstowych. W większości klasycznych zastosowań *data mining* reprezentacja obiektów narzuca się niejako sama – jest to zbiór atrybutów, tworzących krotkę w bazie danych.

Dla dokumentów tekstowych wybór zbioru atrybutów nie jest oczywisty. Dokument tekstowy, będący ciągiem słów, wymaga przekształcenia do postaci, która może być przedmiotem bezpośredniej obróbki przez algorytm. Innymi słowy, niezbędna jest pewna reprezentacja tekstu, która opisywać będzie te parametry dokumentu tekstowego, które determinować będą jego istotne cechy i jednocześnie pozwoli na łatwe porównywanie dokumentów lub ich części. W najczęściej stosowanych metodach stosuje się podejście tzw. „**worka słów**" (ang. ***bag of words***). Polega ono na traktowaniu częstości występowania poszczególnych słów w dokumencie jako atrybutów. Pomija się wówczas miejsce i kolejność występowania słów, chociaż są wyjątki od tej zasady. Poważnym problemem jest wielowymiarowość tak stworzonej przestrzeni atrybutów. Typowe teksty zawierają bowiem od kilku do kilkudziesięciu tysięcy różnych słów. Z tego też powodu, poszukuje się metod wybierania tych słów, czy też grup słów, które są semantycznie najbardziej istotne dla danego zbioru dokumentów lub też dokonuje się wstępnego łączenia słów o pokrewnym znaczeniu w klasy, np. wykorzystując tezaurusy.

### 2.1.1. Reprezentacja unigramowa

Reprezentacją najprostszą, a jednocześnie jak do tej pory powszechnie stosowaną, jest tak zwana reprezentacja unigramowa. Oparta jest ona na zliczaniu występowania poszczególnych słów w treści dokumentu. Częstości te wykorzystywane są następnie do zbudowania wektora reprezentującego dany dokument. Wyróżniamy tu dwa podejścia:

- reprezentację binarną – zaznaczanie wystąpienia lub nie danego słowa,
- reprezentację częstotliwościową – przechowywanie liczby wystąpień słowa w dokumencie.

### 2.1.2. Reprezentacja n-gramowa

W przypadku języków fleksyjnych (np. jęz. polski) kolejność występowania wyrazów w zdaniu może się zmieniać przy zachowaniu tego samego znaczenia, co zresztą często jest wykorzystywane przez poetów, np. w „Stepach Akermańskich" A. Mickiewicza czytamy:

*„Wpłynąłem na suchego przestwór oceanu [...]"*

zamiast

*„Wpłynąłem na przestwór suchego oceanu"*

Jednak odmiana wyrazów pozwala nam bezbłędnie określić funkcję i znaczenie każdego z wyrazów. Tak więc pomijanie w reprezentacji informacji o pozycji słów wydaje się mało szkodliwe dla wyników. Jednak w językach pozycyjnych, jak np. jęz. angielski, pozycja wyrazu w zdaniu ma bardzo duże znaczenie. Ten sam wyraz w zależności od pozycji może mieć różne znaczenie i pełnić różną funkcję, np.

*„Hand me this pen"*

*"Rise your hand"*

Wyrażenia składające się z kilku wyrazów mogą wnosić dodatkową informację, której nie mają one pojedynczo. Dotyczy to często idiomów, np. ruszyć głową. Wydaje się więc cenne przechowywanie nie częstości występowania pojedynczych wyrazów, lecz par, trójek itp., czyli tzw. *n*-gramów.

Do implementacji takich reprezentacji używa się zwykle *n*-wymiarowych macierzy następstwa. Powoduje to duże problemy dla większych wartości *n*, gdyż zapotrzebowanie na pamięć wzrasta wykładniczo. Z tego powodu w praktyce nie używa się reprezentacji większych niż trigramowa.

### 2.1.3. Reprezentacja γ-gramowa

Ograniczenia wynikające ze stałej wartości *n* w reprezentacjach *n*-gramowych spowodowały, że została zaproponowana reprezentacja γ-gramowa [Gawrysiak, 2001]. Polega ona na przechowywaniu informacji o częstości występowania sekwencji o zmiennej długości, przy czym informacji o sekwencjach dłuższych powinno być odpowiednio mniej, niż informacji o sekwencjach krótszych. W kolejnych krokach

tworzona jest reprezentacja unigramowa. Następnie rozszerzana jest o wystarczająco częste sekwencje dwuwyrazowe itd. Podobne koncepcje były wykorzystywane w metodach eksploracji danych do wykrywania zbiorów częstych (Algorytm Apriori) [Agrawal, 1996].

### 2.1.4. Reprezentacja pozycyjna

Kolejnym pomysłem rozbudowującym możliwości reprezentacji jest przechowywanie informacji o pozycjach wystąpienia słów w dokumencie. Zwykle stosowanym podejściem jest podział dokumentu na przedziały i oddzielne zliczanie częstości występowania wyrazów w przedziałach. W ten sposób zamiast informacji o częstości występowania słowa w dokumencie uzyskujemy diagram rozkładu częstości. W ten sposób możemy uwzględniać w pewnym stopniu również układ dokumentu.

## *2.2. Metody przekształcania reprezentacji*

### 2.2.1. Prawo Zipfa

Dodatkowym utrudnieniem w tworzeniu reprezentacji dobrze odzwierciedlających cechy danego dokumentu jest bardzo nierównomierny rozkład częstości występowania poszczególnych słów. Rozkład częstości jest zwykle bardzo charakterystyczny – jest stosunkowo niewiele słów, które bardzo często pojawiają się w treści dokumentu, oraz dużo słów, które występują bardzo rzadko. Słowa występujące bardzo często to w większości zaimki, przyimki i temu podobne części mowy, które samodzielnie nie przekazują praktycznie żadnej informacji. Taką własność rzeczywistych tekstów w języku naturalnym potwierdziły badania Zipfa [Zipf, 1949] i Mandelbrota [Mandelbrot, 1954]. Co więcej okazało się, że rozkład ten jest w zasadzie uniwersalny i niezależny od języka. Do opisu rozkładu Zipf proponuje *dystrybucję Zipfa*:

$$f = \frac{k}{r}$$

udoskonaloną następnie przez Mandelbrota do postaci:

$$f = P \cdot (r + \rho)^{-\beta}$$

gdzie: $f$ – częstość występowania słowa, $r$ – numer kolejny słowa na liście słów uporządkowanej według wzrastającej częstości, zaś $k$, $P$, $\rho$, $\beta$ to parametry, które muszą być dobrane eksperymentalnie. Określają one strukturę słownictwa danego zbioru dokumentów.

### 2.2.2. Zmniejszanie reprezentacji

Powyższe własności dokumentów tekstowych mają bardzo duże znaczenie podczas konstruowania implementacji algorytmów oraz do przekształcania reprezentacji. Słowa o bardzo dużej częstości występowania, nie niosące użytecznej informacji, mogą być usuwane na początku przetwarzania w oparciu o tzw. ***stopword list*** (lub ***stoplist***). Jest to lista zawierająca przyimki, zaimki, rodzajniki itp. Musi być ona tworzona oddzielnie dla każdego języka. Oczywiście mechaniczne usuwanie słów jest ryzykowne, gdyż wiąże się z utratą informacji. Przykładowo wpisanie w wyszukiwarce Google kwerendy „to be or not to be" zostanie ograniczone do „not" (cytat):

> Słowa **to be to be** są bardzo popularne i dlatego zostały zignorowane podczas szukania.
> Wyraz **"or"** został zignorowany w przeszukiwaniu.

Zmniejszanie reprezentacji pozwala nie tylko zwiększyć efektywność przetwarzania, lecz czasami pozwala również poprawić rezultaty grupowania poprzez usuwanie szumu informacyjnego.

Często stosowanym rozwiązaniem jest również usuwanie elementów reprezentacji (np. słów) na podstawie miar statystycznych, czyli tzw. ***pruning***. Usuwanie może obejmować słowa zbyt często występujące lub też występujące w wielu dokumentach.

Również częstym rozwiązaniem jest usuwanie słów bardzo rzadko występujących. Określenie minimalnej wartości progowej powoduje, że rozmiar reprezentacji znacznie się zmniejsza, poprawiając efektywność, a nie wpływając znacząco na wyniki działania algorytmów TDM.

### 2.2.3. Obliczanie wag

W większości zastosowań nie używa się bezpośrednio liczby wystąpień słów. Zamiast nich obliczane są specjalne współczynniki określające ważność słów. Często spotykaną miarą jest ***TF-IDF***. Żeby ją obliczyć, należy najpierw znaleźć miary:

- ***Term frequency*** – *TF(w, d)* oznaczającą liczbę wystąpień słowa *w* w dokumencie *d*.
- ***Document frequency*** – *DF(w)* oznaczającą liczbę dokumentów, w których wystąpiło słowo *w*
- ***Inverse document frequency*** – $IDF(w) = log(|D|/DF(w))$, gdzie $|D|$ to liczba wszystkich rozpatrywanych dokumentów. Miara ta jest tym większa, w im mniejszej liczbie dokumentów występuje słowo *w*.

Jako wagę dla słowa *w* w dokumencie *d* przyjmuje się:

$$d_i = TF(w_i, d) \bullet IDF(w_i)$$

Dzięki przyjęciu takiej metody waga słowa w dokumencie jest duża, jeśli słowo występuje w nim często. Z drugiej strony waga słowa maleje, im częściej słowo występuje w innych dokumentach. Jeśli słowo ma w dokumencie dużą wagę, to oznacza, że dobrze opisuje ten dokument i będzie przydatne przy porównywaniu go z innymi dokumentami.

## 2.3. Algorytmy klasyfikacji dokumentów

W literaturze zostały zaproponowane różnorodne algorytmy klasyfikacji. W wielu wypadkach zostały one zaadaptowane bezpośrednio z technik eksploracji danych. Jednak ze względu na specyfikę dziedziny (bardzo duża liczba atrybutów) nie wszystkie algorytmy mogą być zastosowane wprost. Do najczęściej używanych algorytmów można zaliczyć naiwny klasyfikator Bayesowski, a także algorytmy Rocchio i kNN. Niniejszy rozdział przedstawia skrócony przegląd tych algorytmów w celu pokazania ich zasady działania. Pełniejsze opisy można znaleźć w podanych pozycjach literaturowych. Przegląd algorytmów można znaleźć m.in. w [Gawrysiak, 2001] oraz [Yang, 1998]. Proponowane były również liczne modyfikacje tych algorytmów, jak np. algorytm łączący cechy Rocchio i kNN [Guo, 2003a].

### 2.3.1. Naive Bayes

Algorytm ten bazuje na rachunku prawdopodobieństwa. Klasyfikator określa prawdopodobieństwo przynależności dokumentu do każdej z klas, po czym określenie kategorii dokumentu polega na wybraniu klasy, dla której prawdopodobieństwo to było największe.

Algorytm opiera się o prawo Bayesa:

$$\Pr(a \mid b) = \frac{\Pr(a) \cdot \Pr(b \mid a)}{\Pr(b)},$$

które na potrzeby klasyfikacji przyjmuje postać

$$\Pr(c(x) = d \mid \langle v_1 = x_1, \cdots, v_n = x_n \rangle) = \frac{\Pr(c(x) = d) \cdot \Pr(\langle v_1 = x_1, \cdots, v_n = x_n \rangle \mid c(x) = d)}{\Pr(\langle v_1 = x_1, \cdots, v_n = x_n \rangle)},$$

gdzie:

$d$ - kategoria

$c(x)$ - estymowana kategoria dokumentu x

$\langle v_1 = x_1, \cdots, v_n = x_n \rangle$ - wektor atrybutów ($v_i$ - kolejny atrybut; $x_i$ - wartość atrybutu i dla dokumentu x)

Ponieważ mianownik w twierdzeniu Bayesa nie jest zależny od kategorii, zatem pominięcie go przy obliczeniach nie zmieni rozkładu prawdopodobieństw przynależności dokumentu do kategorii.

Naiwny klasyfikator Bayesowski charakteryzuje się tym, że zakłada brak zależności pomiędzy atrybutami dokumentu względem kategorii. To założenie zwykle nie jest spełnione i stąd wywodzi się nazwa algorytmu. Pomimo tej nieścisłości algorytm zwykle daje zadowalające wyniki. Dzięki temu podejściu następuje znaczne uproszczenie problemu:

$$\Pr(\langle v_1 = x_1, \cdots, v_n = x_n \rangle \mid c(x) = d) = \prod_{i=1}^{n} \Pr(\langle v_i = x_i \rangle \mid c(x) = d)$$

### 2.3.2. Rocchio

Algorytm Rocchio [Rocchio, 1971] klasyfikuje dokument starając się znaleźć kategorię jemu najbliższą w przestrzeni n-wymiarowej.
W tym celu początkowo należy utworzyć wektor dla każdej kategorii na podstawie reprezentacji wektorowej dokumentów, które do niej należą.

$$d_{k,i} = \alpha \cdot \sum_{v \in R_k} \frac{\langle v_i \rangle}{|R_k|} - \beta \cdot \sum_{v \in N_k} \frac{\langle v_i \rangle}{|N_k|}$$

, gdzie:

$d_{k,i}$ - i-ta składowa wektora dla kategorii k

$v_i$ - i-ta składowa wektora dokumentu ze zbioru trenującego

$R_k, N_k$ - zbiory przykładów relewantnych i nieistotnych względem kategorii k

$\alpha, \beta$ - parametry określające istotność przykładów relewantnych i nie

Czasami algorytm ten bywa stosowany w wersji uproszczonej, gdzie $\alpha = 1, \beta = 0$. Wówczas wzór redukuje się do:

$$d_{k,i} = \sum_{v \in R_k} \frac{\langle v_i \rangle}{|R_k|},$$

Co jest równoznaczne wyznaczaniu wektora klasy poprzez uśrednianie wektorów dokumentów ze zbioru trenującego, które należą do danej klasy.

Tworząc wektor zgodnie z metodą Rocchio klasyfikator stara się wyłonić cechy charakterystyczne dokumentów dla danej kategorii i odrzucić te, które określają dokumenty nieistotne względem kategorii.

Klasyfikacja przy pomocy algorytmu Rocchio polega na znalezieniu kategorii, której odległość (względem wybranej miary odległości) w przestrzeni wielowymiarowej do klasyfikowanego dokumentu jest najmniejsza.

Proponowane były liczne rozszerzenia i udoskonalenia tego algorytmu, m.in. w [Joachims, 1996], [Schapire, 1998].

### 2.3.3. kNN

Zasada działania algorytmu kNN sprowadza się do wyszukania w zbiorze trenujących *k* najbardziej podobnych dokumentów do klasyfikowanego i przypisanie mu kategorii takiej, jaką ma większość znalezionych. Znajdowanie dokumentów najbliższych wykonywane jest względem wybranej miary odległości $sim(x, v_k)$ pomiędzy dwoma wektorami w przestrzeni n-wymiarowej.

Jeśli wśród *k* wybranych najbardziej podobnych są dokumenty różnych kategorii, to należy rozstrzygnąć, do której z nich należy klasyfikowany tekst przypisać. Dla każdej z kategorii określane jest średnie podobieństwo pomiędzy badanym dokumentem, a kategorią, do której należą dokumenty wybrane jako najbliższe.

$$sim(x, n_k) = \sum_{v_k \in n_k} \frac{sim(x, v_k)}{|n_k|}$$

, gdzie:

$x$ - wektor klasyfikowanego tekstu

$n_k$ - określa kategorię k wśród najbliższych dokumentów $x$

$v_k$ - wektor należący do zbioru określonego przez $n_k$

Rozszerzenia tego algorytmu były proponowane m.in. w [Guo, 2003] [Guo, 2004].

## 2.4. Algorytmy grupowania dokumentów

Istnieje wiele metod grupowania dokumentów tekstowych różniących się sposobem reprezentacji zbiorów czy też wybranym algorytmem. Wybór metody optymalnej zależy w dużym stopniu od zastosowania, charakterystyki danych oraz spodziewanych wyników.

Algorytmy grupowania można podzielić na hierarchiczne (tworzące hierarchię grup) i niehierarchiczne, np. k-Means. Są dwie najbardziej popularne metody używane w algorytmach hierarchicznych. Jedna polega na wiązaniu ze sobą w kolejnych iteracjach algorytmu kolejnych grup (ang. ***Agglomerative algorithms***), druga na rozbijaniu dużych(ej) grup(y) na coraz to mniejsze (ang. ***Divisive algorithms*** ). Obydwie klasy algorytmów hierarchicznych wymagają stosunkowo długotrwałego przetwarzania. Jednak są bardzo cenne w przypadku, gdy potrzebna jest hierarchiczna struktura grup. Ze względu na potrzebę uzyskania dużej efektywności najczęściej wykorzystywana jest reprezentacja unigramowa w połączeniu z algorytmem *K-Means*.

Grupowanie dokumentów znajduje szczególnie szerokie zastosowanie w systemach wyszukiwania informacji, gdzie wykorzystywane jest do porządkowania i wizualizacji zbiorów odnalezionych dokumentów. Sposób wykorzystania metod grupowania w ogólnodostępnych systemach, szczególnie zintegrowanych z siecią Internet, jest jednak kwestią kontrowersyjną. Nie brak nawet opinii [Banerjee, 1998] o niemożliwości stworzenia uniwersalnego systemu automatycznego grupowania dokumentów takich, jak strony WWW. Niemniej jednak próby wciąż trwają, czego przykładem jest system Carrot stworzony na Politechnice Poznańskiej, rozszerzający możliwości wyszukiwarki Google (www.google.com) o funkcje grupowania dokumentów.

Poniżej znajduje się przegląd najpopularniejszych algorytmów grupowania oraz porównanie ich cech. Więcej informacji na ten temat można znaleźć m.in. w [Ester, 1996], [Hinneburg, 1999], [Milenova, 2002], [Steinbach, 2000].

### 2.4.1. Agglomerative Hierarchical

Algorytm ten polega na wiązaniu w kolejnych krokach małych grup dokumentów. Zasada jego działania jest stosunkowo prosta. Na początku tworzy tyle grup ile jest dokumentów i w każdym następnym kroku łączy dwie najbliższe grupy. Iteracja jest powtarzana dopóki nie otrzymamy takiej liczby grup, jakiej oczekujemy.

W skrócie algorytm wygląda następująco:

1. Obliczenie podobieństwa pomiędzy każdą z par grup i zapisanie ich do macierzy podobieństw grup.
2. Połączenie dwóch grup o największym podobieństwie
3. Zaktualizowanie macierzy podobieństw.
4. Jeśli grup jest więcej niż oczekiwaliśmy to wykonujemy ponownie kroki 2,3,4, w przeciwnym wypadku koniec algorytmu.

Dokumenty w algorytmie są podawane jako wektory. Pomiędzy tymi dokumentami wyliczamy podobieństwo i możemy to zrobić na kilka sposobów, najpopularniejszym jest jednak wyznaczenie cosinusa pomiędzy wektorami:

$$cos(\boldsymbol{d}_1, \boldsymbol{d}_2) = (\boldsymbol{d}_1 * \boldsymbol{d}_2) / ||\boldsymbol{d}_1|| \, ||\boldsymbol{d}_2||$$

Na różne sposoby można też obliczać podobieństwo pomiędzy grupami, które jest podstawą do decyzji łączenia dwóch grup:

1. Pojedyncze powiązanie.
   $d(C1, C2) = min|p1 - p2|; p1 \in C1, p2 \in C2$
2. Kompletne powiązanie.
   $d(C1, C2) = max|p1 - p2|; p1 \in C1, p2 \in C2$
3. Przeciętne powiązanie.
   $d(C1, C2) = 1/(n1*n2)*sum(|p1 - p2|); p1 \in C1, p2 \in C2$

Cechy algorytmu:

1. długi czas działania,
2. dosyć dobra entropia otrzymanych grup,
3. prosty w implementacji.

### 2.4.2. Divisive Hierarchical

Jest to jedna z metod hierarchicznego grupowania danych, ale tworzy hierarchię w porządku odwrotnym, tzn. poprzez dzielenie, a nie łączenie zbiorów.

Początkowo jest tylko jedna grupa zawierająca wszystkie dokumenty. W każdym kolejnym kroku grupa licząca najwięcej elementów jest rozbijana na dwie mniejsze, aż do osiągnięcia odpowiedniej liczby grup.

W pierwszym kroku metody *agglomerative*, jest $\frac{n(n-1)}{2}$ wszystkich kombinacji dwóch dokumentów. W metodzie *divisive* takich kombinacji jest $2^{n-1}-1$. Jest to więcej niż w metodzie *agglomerative*, przez co wzrasta złożoność algorytmu, a co za tym idzie czas wykonania.

Aby ustrzec się przed sprawdzaniem wszystkich przypadków możliwe jest zastosowanie algorytmu opisanego poniżej:

1. Znajdywany jest obiekt z najmniejszym średnim współczynnikiem podobieństwa do pozostałych obiektów. Obiekt ten tworzy nową grupę tymczasową.
2. Dla każdego obiektu i na zewnątrz grupy tymczasowej obliczane są współczynniki:

$D_i$ = [średnia $d(i,j)$ $j \notin$ grupa tymczasowa ] - [średnia $d(i,j)$ $j \in$ grupa tymczasowa]

3. Znajdywany jest dokument *h* z największą różnicą $D_h$. Jeżeli $D_h$ jest dodatnia, to wówczas dokument h jest dostatecznie blisko, aby dołączyć go do grupy tymczasowej.

4. Powtórzenie kroków 2 i 3 dopóki wszystkie różnice $D_i$ będą ujemne. Wtedy grupa jest dzielona na tej podstawie na dwie części (jedna to grupa tymczasowa, a druga to pozostałe dokumenty).
5. Należy wziąć grupę z największą średnią, która jest średnim współczynnikiem podobieństwa do pozostałych obiektów, po czym znowu rozbić ją na dwie grupy zgodnie z krokami 1,2,3.
6. Powtarzane są wszystkie punkty dopóki nie zostanie osiągnięta wymagana liczba grup.

W porównaniu z algorytmem agglomerative, ten algorytm jest trochę mniej efektywny, ale po optymalizacji czas wykonania jest nieco krótszy.

### 2.4.3. COBWEB

Jest to algorytm, którego wynikiem nie jest „płaska" struktura grup, tylko całe drzewo stanowiące grupowanie, którego liście stanowią ostateczną reprezentację grup. Jego działanie opiera się na dodawaniu kolejnych wektorów do drzewa (poczynając od korzenia i schodząc w głąb drzewa), w taki sposób, że struktura drzewa grupowania jest jednocześnie zmieniana. To do którego węzła drzewa jest wektor dodawany i w jaki sposób jest modyfikowana struktura drzewa zależy od funkcji oceny grupowania.

#### Reprezentacja grupowania

Tak jak napisano powyżej grupowanie stanowi drzewo będące wynikiem działania algorytmu. W drzewie tym węzły przechowują rozkłady prawdopodobieństw wszystkich atrybutów – obliczane na podstawie wektorów, które dany węzeł zawiera. Jeśli węzeł posiada węzły potomne oznacza to, że grupa jaką on opisuje, na poziomie o jeden niższym jest rozbita na kilka grup reprezentowanych właśnie przez jego potomków. Gdy węzeł nie posiada potomków (jest liściem drzewa) oznacza to, że jest to grupa „niepodzielna" – zawierające na tyle podobne przykłady, że jej dalszy podział nie jest opłacalny. Idąc z kolei w drugą stronę i dochodząc do korzenia warto zauważyć, że korzeń przechowuje rozkłady prawdopodobieństw wartości atrybutów dla wszystkich wektorów, jakie były przez algorytm przetwarzane.

Reprezentacja rozkładu prawdopodobieństwa dla atrybutów dyskretnych polega na przechowywaniu listy występujących wartości wraz z licznikami wystąpień tych wartości. Dla atrybutów ciągłych natomiast można przechowywać parametry rozkładu (np. dla rozkładu normalnego odchylenie standardowe i wartość oczekiwaną) – będące parametrami funkcji gęstości dla założonego rozkładu.

### Funkcja oceny

Funkcja oceny stosowana w tym algorytmie mierzy przydatność grupowania. Jej wartość jest pomiarem jedynie na poziomie jednego węzła rodzica i jego węzłów potomnych. Sam pomiar polega to na ocenie ile zyskujemy z podziału węzła na węzły potomne. Wykorzystywana jest ona przy wyborze, jaką operację na strukturze drzewa należy wykonać – wybierana jest oczywiście taka modyfikacja, która prowadzi do najlepszego grupowania.

Dokładny wzór na wartość tej funkcji wygląda tak:

$$\vartheta(h) = \frac{1}{|C_h|}\sum_{d\in C_h}\Pr_{x\in\Omega}(h(x)=d)\bullet\left[\sum_{i=1}^{n}\sum_{v\in A_i}\Pr_{x\in\Omega}(a_i(x)=v\,|\,h(x)=d)^2 - \right.$$
$$\left. - \sum_{i=1}^{n}\sum_{v\in A_i}\Pr_{x\in\Omega}(a_i(x)=v)^2\right]$$

gdzie:

- $x$ - rozpatrywany wektor;
- $d$ - kategoria;
- $h(x)$ - kategoria, która wynika z grupowania;
- $C_h$ - liczba kategorii (grup);
- $\Omega$ - rozkład prawdopodobieństwa;
- $i$ - numer atrybutu;
- $A_i$ - zbiór wartości $i$-tego atrybutu;
- $v$ - wartość atrybutu;
- $a_i(x)$ - wartość $i$-tego atrybutu;

### Operacje modyfikacji drzewa

Aby tworzone drzewo grupowania mogło się zmieniać potrzebne są operacje modyfikacji struktury drzewa. Rozważanie jaką operację należy zastosować odbywa się w kontekście jednego węzła (do którego wektor jest dodawany) i jego węzłów potomnych. Możliwe są następujące operacje:

- *Dodanie wektora do istniejącego węzła* – jest to najprostsza z operacji, gdyż polega ona na prostej modyfikacji rozkładów prawdopodobieństw przechowywanych dla każdego atrybutu węzła; to do którego potomka wektor zostanie dodany zależy od tego, które z uzyskanych w ten sposób grupowań jest najlepsze (wg funkcji oceny); w celu stwierdzenia tego faktu wektor jest chwilowo dodawany do każdego potomka oddzielnie i sprawdzana jest wartość funkcji oceny;
- *Utworzenie nowego węzła* – operacja ta polega na za stworzeniu nowego węzła zawierającego tylko rozważany wektor;
- *Podział węzła* – polega na wykonaniu takiej operacji jak zastąpienie węzła przez jego węzły potomne (co właśnie stanowi jego podział); ta operacja jest rozważana tylko w kontekście najlepszego potomka bieżącego węzła rodzica;
- *Połączenie dwóch węzłów* – odbywa się po wybraniu dwóch najlepszych potomków dla wektora; polega na połączeniu tych dwóch węzłów w jeden oraz przeliczeniu wartości rozkładów prawdopodobieństw i przypisaniu potomków do powstałego węzła;

Do prawidłowego działania algorytmu wystarczają już pierwsze dwie z przedstawionych operacji, jednak stosuje też pozostałe ze względu na lepsze własności powstającego drzewa. Jest ono wtedy zdecydowanie mniej uzależnione od kolejności, w jakiej algorytm przetwarzał wektory.

### Działanie algorytmu

Po opisaniu reprezentacji drzewa, funkcji oceny i operacji na drzewie przedstawiona strategia sterowania może się wydawać bardzo prosta. Poniżej przedstawiono atomową cząstkę tej strategii, która to cząstka jest wykonywana rekurencyjnie:

> *Jeśli rozważany węzeł jest liściem, to możliwe są dwie drogi – albo zwiększyć liczbę wektorów przechowywanych w węźle, albo dla tego wektora stworzyć nowy węzeł będący potomkiem bieżącego.*

Jeżeli rozważany węzeł nie jest liściem, to wektor dodawany jest do węzła, a następnie funkcja oceny decyduje o tym, jaka operacja będzie na bieżącym poddrzewie wykonana. Może to być jedna z wyżej opisanych. Po modyfikacji struktury wywoływany jest opisany proces od nowa (tylko zmienia się węzeł, od którego wektor będzie dodawany), aż do osiągnięcia najniższego poziomu drzewa (czyli dotarcia do jakiegoś liścia).

Jak widać z powyższego opisu wektor tyle razy może zmienić strukturę drzewa, ile jest poziomów w drzewie, gdyż na każdym poziomie drzewa podejmowana jest decyzja, co należy z nim zrobić i wektor przechodzi na poziom niżej.

Cechy algorytmu

- Przetwarza wektory danych w trybie przyrostowym – dobrze nadaje się wszędzie tam, gdzie dane napływają z biegiem czasu, lub są to na tyle duże zbiory, że należy je przetwarzać w mniejszych paczkach;
- Nie ma fazy „uczenia się” w algorytmie – już nawet jeden wektor stanowi grupowanie; po dodaniu każdego wektora danych można „odczytać” grupowanie;
- Opis grup jest probabilistyczny – jest to jego duża zaleta, gdyż opis probabilistyczny dobrze opisuje grupę;
- Tworzy grupowanie hierarchiczne – drzewo grupowania – co jest czytelnym i bardzo dobrze zrozumiałym przez człowieka sposobem reprezentacji wiedzy (w tym grupowania);

- Przyjmuje dowolne typy atrybutów (dyskretne i ciągłe), dobrze operuje na mieszanych przestrzeniach atrybutów (ze względu na probabilistyczny opis grup) – co jest rzadkością wśród wszystkich algorytmów klasteryzacji;
- Swobodnie dobiera liczbę grup – nie ma konieczności ustalania z góry liczby grup – można przypuszczać, że w ten sposób uzyskuje on bardzo dużą elastyczność w dostosowywaniu się do danych;
- Jest odporny na brakujące wartości atrybutów – gdy pojawi się brakująca wartość atrybutu w wektorze, to można na podstawie przynależności wektora do grupy wnioskować o wartości tego parametru;

### 2.4.4. K-Means

Algorytm K-Means jest przykładem algorytmu grupowania przez dzielenie. Jeśli *k* jest pożądaną liczbą grup, grupowanie przez dzielenie umożliwia typowo znalezienie od razu *k* grup.

Omawiając algorytm K-Means należy wprowadzić pojęcie centroidu. Centroid jest średnią lub medianą grupy punktów.

Algorytm ten w podstawowej wersji przedstawiony jest poniżej:

1. Wybranie *k* punktów, będących początkowymi centroidami grup.
2. Przyporządkowanie wszystkich punktów do najbliżej leżących centroidów (do klastrów, których centroidy leżą najbliżej).
3. Ponowne obliczenie centroidów na podstawie przyporządkowanych punktów.
4. Powtarzanie kroków 2 i 3 do momentu, aż centroidy (a zatem elementy klastrów) przestaną się zmieniać.

#### Punkty początkowe

Jakość wyznaczonych grup oraz szybkość działania algorytmu K-Means znacząco zależy od punktów początkowych, które są wyznaczane w pierwszej fazie działania algorytmu. A zatem występuje tutaj parametryzacja właściwego algorytmu, zbiorem punktów pełniących rolę początkowych centroidów.

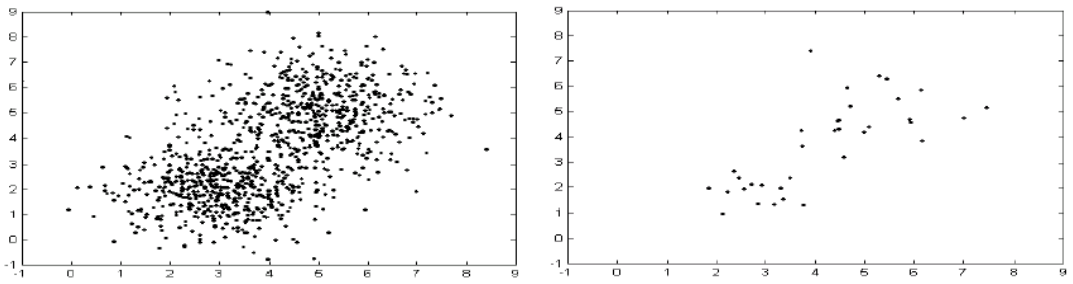
W przypadku, gdy punkty początkowe zostaną wyznaczone w pobliżu rzeczywistych wartości centroidów, zmniejsza się ilość iteracji, niezbędnych do wyznaczenia końcowego rozwiązania, niż w przypadku, gdy punkty początkowe, są zupełnie losowe.

Klasycznym rozwiązaniem wyznaczania punktów początkowych, jest losowanie k dokumentów (gdzie *k* – liczba klastrów, które chcemy wyznaczyć), spośród wszystkich dokumentów i potraktowanie ich jako centroidów.

Proponowanym usprawnieniem jest określenie maksimów łącznej gęstości prawdopodobieństwa, występowania danych i umieszczenie centroidów w ich pobliżu.

Zatem dobrą propozycją rozwiązania tego problemu jest oszacowanie gęstości i odnalezienie jej maksimów. Wyznaczenie gęstości w przypadku dużej ilości punktów (dokumentów) jest trudne.

Podstawowa heurystyka jest taka, że kilka, kilkanaście próbek danych, naturalnie przybliża rzeczywisty rozkład gęstości.



**Rysunek 2-1 Zbiór wszystkich punktów (po lewej) oraz losowej próbki (po prawej)**

Ta obserwacja wskazuje na to, że rozwiązanie otrzymane z klasteryzacji małej próbki, może być dobrym przybliżeniem rzeczywistych centroidów całego zbioru danych.

### Algorytm ulepszania punktów początkowych

Aby zapobiec losowemu generowaniu punktów początkowych (które mogą generować np. puste grupy), opracowano algorytm klastrowania klastrów.

W pierwszej fazie wybieranych jest *J* losowych próbek $S_i$ *i=1,..,J*, ze zbioru danych. Próbki te są grupowane przy pomocy algorytmu K-Means, przy czym, ewentualne puste klastry są usuwane, a w ich miejsce tworzone nowe. Centroid takiego nowo

utworzonego klastra przyjmuje wartość punktu najbardziej oddalonego od jakiegokolwiek centroidu (spośród wszystkich punktów i centroidów).

Zbiory $CM_i$, $i = 1,...,J$ są rozwiązaniami pierwszej fazy, tworząc zbiór $CM$.

Następnie zbiór $CM$ jest klastrowany przy użyciu klasycznego algorytmu K-Means, inicjalizowanego punktami $CM_i$, produkując rozwiązania $FM_i$.

Jako optymalne punkty początkowe dla całego zbioru danych, wybierane są te spośród $FM_i$, dla których otrzymywany jest najmniejszy wskaźnik zniekształcenia grupy, dla zbioru danych $CM$.

Algorytm można przedstawić w następującej postaci:

**Algorithm Refine**(*SP, Data, K, J*)

0. $CM = \phi$

1. For $i$=1,…,$J$

    a. Niech $S_i$ będzie losowym podzbiorem danych

    b. $CM_i$ = KMeansMod($SP$, $S_i$, $K$)

    c. $CM = CM \cup CM_i$

2. $FMS = \phi$

3. For $i$=1,…,$J$

    a. $FM_i$ = KMeans($CM_i$, $CM$, $K$)

    b. $FMS = FMS \cup FM_i$

4. $FM = ArgMin\{Distortion(FM_i, CM)\}$ po $FM_i$.

5. Return ($FM$)

## 2.5. Ocena jakości klasyfikacji

Aby zmierzyć jakość klasyfikacji można w łatwy sposób policzyć poprawność. Zakładając, że dysponujemy zbiorem dokumentów opisanych prawidłowym przydziałem do grup możemy ten zbiór podzielić na zbiór trenujący i testowy. Podział może być arbitralny lub losowy. Po zakończeniu procesu uczenia klasyfikatora na danych trenujących przeprowadzamy klasyfikację zbioru testowego. Poprawność jest liczona jako proces poprawnie zaklasyfikowanych dokumentów.

$$\text{Poprawność} = \frac{c}{d}, \text{ gdzie}$$

$c$ – liczba prawidłowo zaklasyfikowanych dokumentów

$d$ – liczba wszystkich dokumentów poddanych klasyfikacji

Miara ta zakłada przydział każdego dokumentu do dokładnie jednej klasy. Poprawność może zależeć od wielkości zbioru trenującego i testowego, liczby klas oraz występowania słów ze zbioru testowego w zbiorze trenującym.

## 2.6. Ocena jakości grupowania

Mierzenie jakości grupowania jest trudniejsze niż w przypadku klasyfikacji, gdyż samo zadanie grupowania jest postawione w sposób mniej precyzyjny. Do mierzenia jakości grupowania używane są dwa główne rodzaje miar. Miary pierwszego rodzaju umożliwiają porównanie różnych zbiorów klastrów bez dodatkowych informacji zewnętrznych i są nazywane miarami *wewnętrznej jakości*. Przykładem takiej miary jest „całkowite podobieństwo" oparte na podobieństwie wszystkich par dokumentów w klastrze.

Drugi rodzaj miar pozwala sprawdzić jak dobrze działa grupowanie, przez porównanie wygenerowanych grup do predefiniowanych znanych klas. Ten typ miary nazywany jest miarą *zewnętrznej jakości*. Przykładem tego typu miary jest „poprawność grup", entropia oraz F-measure.

### 2.6.1. Podobieństwo

W przypadku braku zewnętrznych informacji, takich jak przynależność do klas, miarą podobieństwa i jakości klastra może być jego spójność.

Jedną z metod obliczania spójności zbioru klastrów jest użycie całkowitego podobieństwa, ważonego podobieństwami wewnętrznymi poszczególnych klastrów.

Całkowita miara podobieństwa (ang. ***Overall similarity***) obliczana jest przy pomocy następującego wzoru:

$$\frac{1}{|S|^2}\sum_{\substack{d\in S\\ d'\in S}}\cos ine(d',d)$$

### 2.6.2. Poprawność grup

Miara ta została zaproponowana przez autora niniejszej rozprawy i pozwala określić zgodność grup uzyskanych w procesie grupowania z predefiniowanymi klasami, które są definiowane przez ludzkiego eksperta. Miara ta zdefiniowana jest następująco:

Niech $c^i_k$ będzie liczbą dokumentów z predefiniowanej klasy *k*, które znalazły się w utworzonej grupie *i*. Wówczas:

$$c^i_{max} = \max_{\forall k} c^i_k$$

oraz

$$\text{Poprawność grup} = \frac{\sum_i c^i_{\max}}{|D|}$$

gdzie |*D*| jest liczbą wszystkich grupowanych dokumentów. Miara ta ma zawsze wartość dodatnią nie większą niż 1 (100%). Najwyższa wartość tej miary występuje wówczas, gdy utworzone grupy pokrywają się dokładnie z grupami predefiniowanymi.

### 2.6.3. Entropia

Entropia jest miarą jakości grupowania (najlepsza możliwa entropia jest uzyskiwana, gdy w klastrze jest tylko jeden dokument).

Niech CS będzie rozwiązaniem klasteryzacji. W pierwszym kroku wyznaczania entropii, oblicza się rozkład klas w elementach każdego klastra – dla klastra *j* obliczana jest wartość $p_{ij}$, prawdopodobieństwo, że element klastra *j* należy do klasy *i*. Następnie używając tych prawdopodobieństw, entropia każdego klastra obliczana jest przy użyciu następującego wzoru:

$$E_j = -\sum_i p_{ij} \log(p_{ij})$$

przy czym sumowanie odbywa się po wszystkich klasach. Całkowita entropia zbioru klastrów obliczana jest jako suma entropii wszystkich klastrów, ważona wielkością klastrów:

$$E_{CS} = \sum_{j=1}^{m} \frac{n_j * E_j}{n}$$

gdzie $n_j$ jest rozmiarem klastra *j*, *m* jest liczbą klastrów, a *n* liczbą wszystkich klastrowanych dokumentów.

### 2.6.4. F-measure

Drugą zewnętrzną miarą jakości grupowania jest *F-measure*. *F-measure* bardziej nadaje się do oceny algorytmów hierarchicznych.

Miarę tę wyznacza się w następujący sposób: Dla każdego klastra *j* oraz klasy *i* oblicza się następujące wartości:

$$Recall(i, j) = n_{ij}/n_i$$

$$Precision(i, j) = n_{ij}/n_j$$

Gdzie $n_{ij}$ jest liczbą członków klasy *i*, należących do klastra *j*, $n_j$ jest liczbą elementów klastra *j*, a $n_i$ liczbą elementów należących do klasy *i*.

*F-measure* klastra *j* oraz klasy *i* jest dana wzorem:

$$F(i, j) = (2 * Recall(i, j) * Precision(i, j)) / ((Precision(i, j) + Recall(i, j))$$

Całkowita miara *F* wyznaczana jest przy pomocy wzoru:

$$F = \sum_i \frac{n_i}{n} \max\{F(i, j)\}$$

gdzie *n* jest liczbą wszystkich dokumentów.

## 2.7. Sumaryzacja

Sumaryzacja jest procesem streszczania źródłowego tekstu do postaci krótszej, jednak z zachowaniem zawartości informacyjnej lub przynajmniej jej najważniejszych części.

Próby modelowania metod działania ludzi profesjonalnie zajmujących się streszczaniem dokumentów zostały przedstawione w [Endres-Niggermeyer, 1999]. Pośród wielu występujących strategii można przytoczyć kilka przykładowych:

- pomijanie treści trywialnych
- pomijanie fragmentów, które nie wnoszą treści
- pomijanie cytatów, pozostawianie wyłącznie własnych wyników autorów dokumentu
- pozostawianie treści nowych lub oryginalnych
- pozostawianie treści różniących się od reszty

Wymienione powyżej zasady polegają na ekstrakcji treści. Pożądaną cechą streszczeń jest również uogólnianie, które pozwala w krótszym dokumencie wyrazić najważniejsze treści. Kluczowym elementem takiego podejścia jest grupowanie treści z kilku zdań i wyrażanie ich w postaci jednego uogólnionego zdania. Zwykle wymaga to jednak wykorzystania dodatkowej wiedzy z danej dziedziny, przez co jest procesem bardzo złożonym. Możemy przeanalizować to na przykładzie. Rozważmy poniższy tekst źródłowy:

*Wczoraj po południu poszliśmy do ogrodu zoologicznego. Widzieliśmy tam żyrafy, lwy, lamparty, tygrysy, hipopotamy, wielbłądy, gazele i antylopy. Później oglądaliśmy wybiegi dla małp oraz ptaków tropikalnych.*

Powyższy tekst może być przez człowieka streszczony do postaci:

*Wczoraj w zoo widzieliśmy wiele zwierząt.*

W niniejszym przykładzie wykonano kilka ciekawych przekształceń tekstu:

1. Uogólniono informację o czasie zdarzenia (pominięcie frazy *„po południu”*).
2. Określenie *„ogród zoologiczny”* zastąpiono przez krótszy odpowiednik *„zoo”*.

3. Usunięto wyrazy o charakterze łączników: *„tam”*, *„Później”*.
4. Pominięto fragmenty nie wnoszące istotnej informacji (domyślne): *„poszliśmy do”*, *„wybiegi dla”*.
5. Wyrazy o bliskim zakresie znaczeniowym potraktowano wspólnie: *„widzieliśmy”* i *„oglądaliśmy”*.
6. Wszystkie wyrazy należące do jednej grupy znaczeniowej zastąpiono nazwą grupy: *„zwierzęta”*.

Aby uzyskać tego typu streszczenie człowiek wykorzystuje następujące umiejętności:

- znajomość zasad syntaktycznych języka naturalnego, w tym także zasad stylistycznych
- umiejętność rozpoznawania przekazu semantycznego słów i wiązanie ich z obiektami ze świata rzeczywistego
- znajomość dziedziny, której tekst dotyczy
- umiejętność wyobrażenia sobie opisanej sytuacji
- umiejętność oceny przydatności informacji w kontekście zainteresowań potencjalnego odbiorcy.

Z tego wynika, że streszczenie może mieć różną treść w zależności od osoby streszczającej, jej wiedzy i sytuacji, w której ta osoba się znajduje. Często ta sama osoba po upływie jakiegoś czasu tworzy inne streszczenie [Endres-Niggemeyer, 1998].

Jak widać na powyższym przykładzie stosowane strategie są trudne do pełnego zamodelowania i często wymagają głębokiej analizy i zrozumienia tekstu. Stosowane w praktyce metody automatyczne są zwykle pewnym przybliżeniem strategii wykorzystywanych przez ludzi.

### 2.7.1. Podział metod sumaryzacji

W pracach poruszających temat sumaryzacji proponowane są różnorakie rozwiązania i podejścia w zależności od stosowanych metod, a także od przewidywanych zastosowań. Inne rozwiązania są przydatne do tworzenia analiz przeglądowych w dziedzinach naukowych, a inne w przypadku tworzenia krótkich notatek podsumowujących ogólną

tematykę dokumentu tekstowego. Tworzenie pełnowartościowych streszczeń tekstów z dowolnej dziedziny nadal pozostaje zagadnieniem bardzo trudnym, gdyż wymaga pełnego zrozumienia tekstu. Tworzenie krótkich podsumowań, które pozwalają ocenić czy całość tekstu jest warta przeczytania jest oczywiście łatwiejsze do wykonania.

#### Ze względu na rodzaj danych wejściowych

Sumaryzacja może obejmować:

- pojedynczy dokument, przy czym w procesie mogą być wykorzystane inne dokumenty z danej dziedziny jako kontekst, np. w celu obliczania miar statystycznych;
- zbiór dokumentów. W tym przypadku problem sumaryzacji może dotyczyć zarówno tworzenia streszczenia całego zbioru dokumentów lub też wybranego zagadnienia na podstawie zapytania użytkownika. Dodatkowo zbiór dokumentów może być statyczny, jak i zmieniający się w czasie, np. zbiór artykułów z agencji prasowej.

#### Ze względu na rodzaj wyniku

Oczekiwane wyniki sumaryzacji mogą się znacząco różnić w zależności od zastosowań. Wynikiem sumaryzacji może być:

- zbiór słów kluczowych charakteryzujących dany dokument,
- tytuł dokumentu streszczanego wygenerowany na podstawie jego treści,
- dokument tekstowy o objętości mniejszej niż dokument źródłowy.

#### Ze względu na parametry wejściowe

W niektórych praktycznych rozwiązaniach sumaryzacja jest postrzegana jako część procesu IR i służy do prezentacji użytkownikowi wyników przeszukiwania zbioru dokumentów. Użytkownik po podaniu zapytania otrzymuje streszczenia znalezionych dokumentów lub też jedno streszczenie obejmujące treść wszystkich znalezionych dokumentów. Z tego względu można wyróżnić jeszcze jeden podział metod sumaryzacji:

- streszczenia ogólne, których zadaniem jest przedstawienie „najważniejszych" informacji zawartych w treści źródłowej,
- oparte o zapytanie użytkownika (ang. ***query-based***), które mają na celu stworzenie spójnego dokumentu zawierającego informacje pasujące do zapytania, znalezione w zbiorze wejściowym.

### Ze względu na znajomość dziedziny

Możemy wyróżnić następujące przypadki:

- metoda korzysta z wiedzy na temat dziedziny, do której należy tekst poddawany sumaryzacji. W tym przypadku możliwe jest wykorzystywanie specyficznych cech dziedziny, np. słowników, teraurusów, a także dodatkowych reguł, które pozwalają uzyskać lepsze efekty sumaryzacji tekstów należących do dziedziny. Z drugiej strony ogranicza to zakres stosowalności metody.
- metoda nie zna dziedziny i/lub nie korzysta z dodatkowej wiedzy na temat dziedziny. Tego typu metody mogą charakteryzować się niższą skutecznością, jednak są bardziej uniwersalne.

### Ze względu na zasadę działania

Bardzo często w literaturze wyróżnia się dwa podejścia do sumaryzacji:

- sumaryzacja poprzez ekstrakcję (**ekstrakcja, *ang. extraction***) – jest to podejście historycznie starsze. Polega ono na wybieraniu z dokumentu pewnych zdań, które najlepiej oddają jego treść. Innymi słowy z tekstu źródłowego usuwane są pewne fragmenty, które są nadmiarowe lub mało istotne.
- sumaryzacja generacyjna (**abstrakcja, *ang. abstraction***) – polega na tworzeniu streszczenia na podstawie wcześniej przygotowanych i zagregowanych informacji. Tekst źródłowy jest zamieniany na pewną reprezentację semantyczną, a następnie wybierane są z niej najważniejsze elementy. Zdania nie są wybierane z tekstu wejściowego, ale tworzone na jego podstawie. Dzięki czemu dłuższe fragmenty tekstu mogą być zastępowane przez ich krótsze odpowiedniki. W szczególności jedno zdanie wynikowe może zawierać treść wielu zdań z tekstu źródłowego.

Jednak podział ten nie jest do końca ścisły, gdyż w przypadku wielu metod opartych o ekstrakcję stosowane są różnego rodzaju techniki post-processing'u, które mają na celu „wygładzenie" tekstu streszczenia, np. wstawianie zaimków. Z tego powodu streszczenie może zawierać zdania inne niż tekst źródłowy.

Poniższa tabela przedstawia porównanie właściwości obu podejść.

| **Abstrakcja** | **Ekstrakcja** |
|---|---|
| Wyniki bardziej zbliżone do wyników uzyskiwanych przez człowieka. | Wyniki mniej zbliżone do wyników uzyskiwanych przez człowieka. |
| Streszczenie zawiera zdania nie występujące w tekście źródłowym, lecz będące ich uogólnieniem. | Streszczenie zawiera wybrane zdania z tekstu źródłowego. |
| Możliwe zaburzenie przekazu treści w przypadku błędnej interpretacji semantyki tekstu źródłowego. | Znikoma modyfikacja treści. Możliwe pominięcie ważnych treści. |
| Możliwe utracenie stylu pisarskiego autora. | Pozostawienie stylu pisarskiego autora poprzez zachowanie pełnych zdań. |
| Długość zdań w tekście źródłowym nie wpływa znacząco na streszczenie. | Długie zdania w tekście źródłowym zaburzają streszczenie (wydłużają streszczenie lub zmniejszają liczbę zdań w streszczeniu). |
| Złożone algorytmy i długi czas przetwarzania. | Prostsze algorytmy i mniej czasochłonne. Intensywnie wykorzystywane metody statystyczne. |

### 2.7.2. Przegląd algorytmów sumaryzacji

Większość dotychczasowych prac dotyczących sumaryzacji korzysta ze statystycznych metod ekstrakcji zdań. Koncepcja ta została zaproponowana w latach 50-tych [Luhn, 1958] i była intensywnie wykorzystywana przez wiele lat.

Algorytm Luhna ma w zarysie następującą postać:

1. Ze źródłowego ciągu tekstowego pobierane są słowa.
2. Słowa znajdujące się w tabeli słów niepożądanych (ang. ***stop-list***) są usuwane (nie są brane pod uwagę w dalszym przetwarzaniu).
3. Słowa podobne (różniące się mniej niż 7 znakami) są grupowane, co ma na celu łączenie słów bliskoznacznych. Każda grupa słów posiada atrybut częstość występowania, który jest równy liczbie wystąpień słów z danej grupy.
4. Usuwane są grupy słów z najmniejszą i z największą częstością występowania (ang. ***pruning***). Pozostałe grupy słów pozostają jako najbardziej znaczące.
5. Dla każdego zdania w tekście wyznaczany jest współczynnik wg algorytmu:
   5.1. Zdanie jest dzielone na substringi, z których każdy zawiera słowo ważne oraz słowa nie ważne z sąsiedztwa,
   5.2. dla każdego substringa jest wyznaczany współczynnik: waga słowa ważnego / ilość słów w substringu,
   5.3. suma współczynników substringów jest sumowana jako globalny współczynnik dla całego zdania,
   5.4. zdania, których globalny współczynnik jest wyższy od założonego znajdą się w wynikowym tekście.

W 1960 roku został zaproponowany algorytm TWR będący rozszerzeniem algorytmu Luhna. Do głównych modyfikacji należały:

- zwiększanie wag słów, które wystąpiły w tytule lub podtytułach dokumentu
- zwiększanie wag zdań występujących na początku i na końcu dokumentu w stosunku do wag słów występujących w środkowej części dokumentu.

W [Edmunson, 1969] została zaproponowana metoda wyboru zdań na podstawie występowania specyficznych fraz, np. „dokument opisuje”, „podsumowując” itp. Algorytm Edmundsona zalicza się do metod klasycznych. Pomimo upływu czasu jest

nadal stosowany w wielu dzisiejszych systemach (z ulepszeniami). W tym algorytmie stosowane są następujące typy reguł do oceny ważności zdań:

1. (*Cue*) podnieś wagę sentencji, jeśli zawiera wyrażenia silnie akcentujące typu: „podsumowując", „w nawiązaniu do", „znacząco", itp. (słowa, które podkreślają wagę słów stojących za nimi).
2. (*Key words*) podnieś wagę sentencji, które zawierają słowa występujące statystycznie najczęściej w badanym tekście. W tym przypadku pomijane są słowna niepożądane występujące na tzw. ***stoplist*.**
3. (*Title*) podnieś wagę sentencji, które zawierają słowa z tytułu badanego tekstu.
4. (*Location*) W zależności od typu badanego tekstu: podnieś wagę sentencji, które występują w określonym miejscu w badanym tekście, np. dla artykułów prasowych będą to sentencje występujące na początku tekstu.

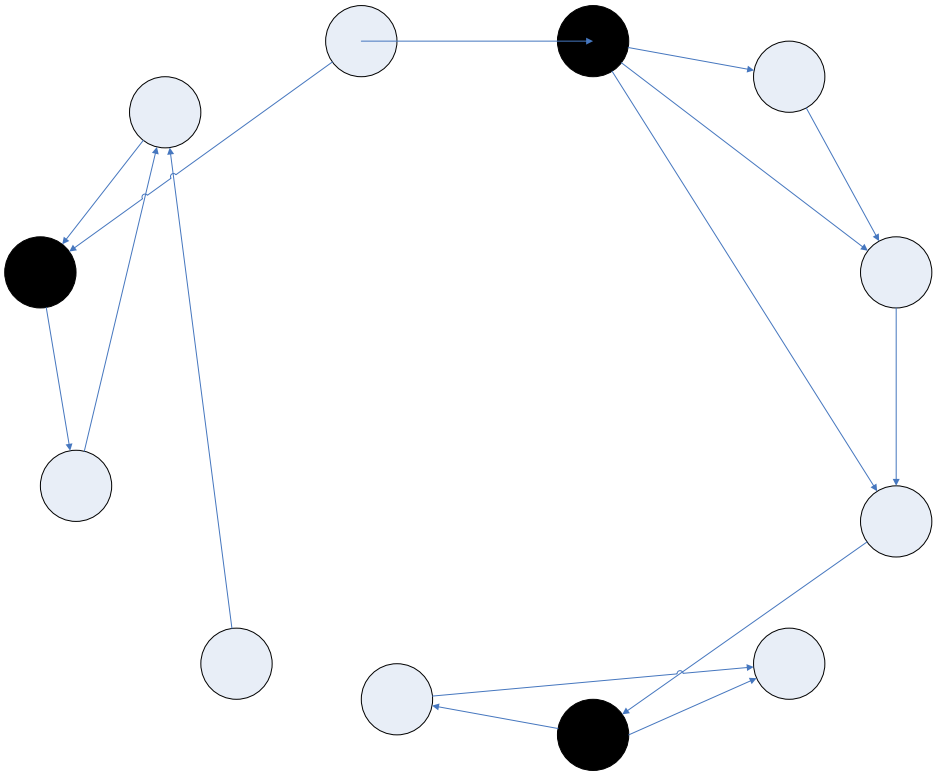
Wpływ wymienionych reguł na ocenę zdań jest określany odpowiednimi wagami o wartościach dobieranych eksperymentalnie.

Algorytm Edmundsona nie zapobiega dobrze powstawaniu redundancji w końcowym streszczeniu. Algorytmy takie jak ***MMR*** (ang. ***Maximum Marginal Relevance***) zapobiegają powstawaniu redundancji.

W literaturze proponowane były również algorytmy bazujące na teorii grafów. Idea opiera się na zbudowaniu grafu sentencji. Przykład takiego algorytmu pochodzący z [Ganapathiraju, 2002] przedstawiono poniżej:

1. Dla każdej sekwencji utwórz węzeł grafu
2. Jeśli dwie sentencje posiadają wspólne słowo - połącz węzły tych sekwencji krawędzią
    a. Jeśli sentencje mają więcej niż jedno wspólne słowo – dla każdej pary utwórz osobną krawędź

Oto przykładowy wynik działania algorytmu:



W wyniku działania algorytmu można zauważyć dwie właściwości grafu:

1. Podgrafy, których węzły nie są połączone krawędziami z innymi podgrafami zawierają sentencje, które dobrze określają jeden temat.
2. Węzły sentencji, które posiadają największą kardynalność (wychodzi z nich najwięcej krawędzi) są sentencjami najbardziej znaczącymi w badanym tekście i powinny być uwzględnione w sumaryzacji.

Wspomniany wcześniej algorytm ***MMR*** (ang. ***Maximum Marginal Relevance***) wykorzystuje zasadę działania grafu sentencji. Graf jest wykorzystywany do szukania i eliminacji podobnych sentencji. Rozważmy dwie przykładowe sentencje:

*W mieście latem jest cicho.*

*W naszym mieście jest dzisiaj bardzo cicho.*

Obie sekwencje posiadają wspólne słowa: *„W”, „mieście”*, *„jest”* i *„cicho”*.

Podobieństwo obu sentencji wyznaczane jest z wzoru:

*Podobieństwo = ilość wspólnych słów / ilość słów w obu sentencjach*

Dla podanego wyżej przykładu podobieństwo wynosi: 4 / 12 = 0,33…

W zależności od przyjętego poziomu podobieństwa jedna z tych sentencji może być odrzucona podczas sumaryzacji, ponieważ niesie ze sobą informację redundantną.

Paice [Paice, 1990] przedstawia przegląd różnych metod sumaryzacji wykorzystujących wybieranie zdań z oryginalnego tekstu. W literaturze występują zarówno algorytmy i systemy mocno zależne od analizowanej dziedziny i korzystające ze specjalistycznej wiedzy [DeJong, 1997], [Tait, 1983], [Jacobs, Rau, 1990], jak i takie, które są niezależne od tematyki analizowanego tekstu, np. NetSumm [Preston, Williams, 1994] czy sumaryzator Xeroxa [Kupiec, 1995]. Xerox używa algorytmu uczącego się, który na podstawie analizy korpusu tekstu określa wagi różnych własności używanych do ekstrakcji zdań, np. częstość słów, użycie konkretnych fraz, słowa tematyczne, pozycje paragrafów, długość zdań czy wielkość pierwszych liter wyrazów.

Proponowane były również rozwiązania bazujące na głębokiej analizie tekstu i na wynikach uzyskiwanych z IE, w szczególności na znajdywaniu i uzupełnianiu wzorców zdefiniowanych na konferencjach MUC. W [McKeown, Radev, 1995], [Radev, 1997] zaproponowano system SUMMONS, który tworzy streszczenie na podstawie uzupełnionych wzorców pozyskanych z serii artykułów prasowych opisujących jedno wydarzenie lub grupę podobnych wydarzeń. System ten tworzy streszczenia na podstawie zgromadzonych faktów w sposób generacyjny. Jednak tego typu rozwiązania zwykle są ograniczone do określonej dziedziny, gdyż bazują na określonym zbiorze wzorców.

W [Nagao, Hasida, 1998] przedstawiona została metoda streszczania dokumentów wzbogaconych o dodatkową informację semantyczną zgodnie ze projektem ***GDA*** (ang. ***Global Documents Annotation***). Jednak notacja GDA wymaga opisywania dokumentów zbiorem odpowiednich tagów przez ich autorów i dlatego zastosowanie tej metody pozostaje ewentualnie sprawą przyszłości. Bardziej praktyczne jest zastosowanie słowników i tezaurusów w celu wykorzystania informacji o relacjach semantycznych wyrazów, np. synonimicznych. Przykładem takiego podejścia jest [Barzilay, Elhadad, 1997]. Zaproponowane tam wykorzystanie angielskiego słownika

WordNet do konstrukcji łańcuchów leksykalnych pozwala poprawić jakość sumaryzacji.

W wielu ośrodkach badawczych trwają również prace nad stworzeniem narzędzi do sumaryzacji tekstów polskich, m.in. [Ciura, Grund, Kulików, 2004], [Suszczanska, Kulików, 2003].

# 3. Model płytkiej analizy tekstu

Niniejszy rozdział prezentuje model płytkiej analizy tekstu (***ang. Shallow Text Processing***, ***STP***). Omówiono kolejno elementy składowe modelu i wiążące się z nimi zagadnienia. Na tle rozwiązań proponowanych w literaturze przedstawiono również wyniki badań prowadzony w ramach projektów badawczych prowadzonych przez autora niniejszej rozprawy. Projekty te wykonano w ramach przedmiotu „Inteligentne Systemy Informacyjne" prowadzonego przez prof. dr hab. inż. Mieczysława Muraszkiewicza na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej w latach 2004-2005. Przedstawiono również zaproponowane algorytmy i metody realizujące wybrane elementy modelu oraz wnioski płynące z przeprowadzonych eksperymentów.

Jak to zostanie przedstawione w kolejnych rozdziałach model STP wraz z narzędziami go realizującymi stanowi dobrą podstawę do rozwiązywania i wspierania rozwiązań praktycznych występujących w wielu zadaniach TDM.

W [Piskorski, 2001] zostały zaproponowane następujące elementy modelu STP:

- tokenizacja,
- wykrywanie końca zdania,
- analiza morfologiczna,
- usuwanie niejednoznaczności,
- wykrywanie nazw własnych,
- zastępowanie zaimków.

W niniejszej pracy przeanalizowano powyższe aspekty płytkiej analizy tekstu w kontekście przetwarzania języka polskiego. Dodatkowo zaproponowano dwa dodatkowe elementy analizy:

- rozkład zdań złożonych na zdania proste,
- rozbiór zdań.

Są one naturalnym rozszerzeniem zakresu analizy i intensywnie korzystają z wcześniej zaproponowanych etapów analizy. Zwiększają także zakres płytkiej analizy tekstu i pozwalają potencjalnie poprawić wyniki realizacji zadań TDM, w szczególności sumaryzacji.

Więcej informacji o środowisku implementacyjnym znajduje się w załączniku A na str. 134.

## 3.1. Tokenizacja

W większości rozwiązań typu STP początkowym etapem przetwarzania dokumentów tekstowych jest przekształcenie ciągu znaków w uporządkowany zbiór elementów zwanych tokenami (ang. ***tokens***). Tokeny mogą odpowiadać słowom, liczbom, znakom interpunkcyjnym, nazwom itp. Oprócz fragmentu tekstu (słowa, znaku interpunkcyjnego) token zawiera dodatkowe atrybuty, jak np. numer kolejny, pozwalający zachować kolejność tokenów. Proces tworzenia tokenów określany jest jako **tokenizacja** (ang. ***tokenization***). Uporządkowany zbiór tokenów służy jako podstawa dalszej analizy leksykalnej. Na kolejnych etapach tokeny są uzupełniane dodatkowymi informacjami, będącymi wynikiem danego etapu analizy. Zwykle jest to realizowane poprzez dodawanie kolejnych atrybutów opisujących token.

Czasami w procesie tokenizacji wykonywane są dodatkowe zadania analizy tekstu, jak np.:

- wykrywanie końca zdania
- łączenie części przeniesionych wyrazów
- rozpoznawanie skrótów
- oddzielanie przedrostków i przyrostków

Zadania te mogą także być wydzielone i przetwarzane na dalszych etapach przetwarzania. Wybrane systemy przeprowadzające proces tokenizacji są przedstawione w [Greffenstette, 2000], [Grover, 2000], [Gillam, 1999]. Do wykonania tego zadania może być wykorzystywany szeroko znany skaner strumienia znaków *lex* oraz *flex* opisany w [Levine, 1992]. W niektórych rozwiązaniach na etapie tokenizacji przeprowadzana jest wstępna klasyfikacji tokenów zgodnie z predefiniowanymi

grupami [Piskorski, 2001], jak np. słowo pisane małymi literami, słowo rozpoczynające się z dużej litery, liczba całkowita, godzina, data, adres URL itp. Klasyfikacja taka jest przeprowadzana bez użycia informacji kontekstowych oraz bez wykorzystywania zaawansowanych technik lingwistycznych. Jednym z rozwiązań jest wykorzystanie wyrażeń regularnych. Tego typu wstępna klasyfikacja może ułatwić dalszą analizę tokenów oraz zwiększyć poziom abstrakcji przy konstruowaniu reguł analizy syntaktycznej.

#### Implementacja

W ramach opracowanej platformy zdefiniowano uniwersalny format zapisu tokenów oparty o XML. Zdefiniowano również rodzaje rozpoznawanych tokenów. Ich pełna lista i opis znajduje się w części Załącznik A: Środowisko implementacyjne (str. 134).

W ramach [Proj2] i [Proj3] zaproponowano i zaimplementowano algorytmy podziału tekstu na tokeny i rozpoznawania rodzaju tokenów. Implementacja [Proj2] opiera się o zbiór klas (Java), z których każda rozpoznaje konkretny rodzaj tokena. W szczególności możliwe jest łączenie tokenów, np. połączenie tokenów „10”, „:”, „30”, które zostały rozpoznane niezależnie w jeden token określający godzinę. Konstrukcja modułu pozwala na tworzenie dodatkowych klas rozpoznających nowe rodzaje tokenów i dołączanie ich do modułu bez rekompilacji. Przeprowadzono testy na wybranych danych rzeczywistych. Po przetworzeniu pliku poprawność została sprawdzona przez operatora. Skuteczność rozpoznawania rodzaju tokena jest bardo wysoka (97%-100%).

Miary poprawności przedstawia poniższa tabela.

| Plik | Liczba tokenów | Liczba tokenów poprawnie rozpoznanych | Poprawność |
|---|---|---|---|
| Architekci | 872 | 871 | ~100% |
| Bankowa superfuzja… | 1316 | 1316 | 100% |
| Chińska nauka | 1122 | 1122 | 100% |
| lalka1 | 1676 | 1676 | 100% |
| MAGNETYZER | 955 | 955 | 100% |
| Mechanizm różnicowy | 390 | 390 | 100% |
| naglowki | 2442 | 2369 | 97% |
| Pierwsze dziecko… | 640 | 640 | 100% |
| Młodzi w Polsce… | 466 | 465 | ~100% |
| Seleukos | 2533 | 2527 | ~100% |
| we_test | 273 | 273 | 100% |

Implementacja [Proj3] opiera się o wyrażenia regularne. Do tego celu zaprojektowano i zaimplementowano własny język opisu wyrażeń regularnych. Dzięki temu rozpoznawanie tokenów i ich rodzaju oraz generowanie danych wyjściowych jest realizowane z wykorzystaniem przygotowanego zbioru wyrażeń. Takie podejście pozwala na szerokie rozszerzanie funkcjonalności.

| Plik | Liczba tokenów | Liczba tokenów poprawnie rozpoznanych | Poprawność |
|---|---|---|---|
| Architekci | 795 | 783 | 98% |
| Bankowa superfuzja… | 1340 | 1313 | 98% |
| lalka1 | 1698 | 1690 | ~100% |
| MAGNETYZER | 961 | 940 | 98% |
| Mechanizm różnicowy | 392 | 390 | 99% |
| naglowki | 2409 | 2391 | 99% |
| Pierwsze dziecko… | 664 | 657 | 99% |
| Młodzi w Polsce… | 469 | 461 | 98% |
| Seleukos | 2582 | 2537 | 98% |
| we_test | 285 | 275 | 96% |

Różnice całkowitej liczby tokenów występujących w pliku występujące w wynikach testów są spowodowane rozbieżnościami w rozpoznawaniu tokenów i ich łączenia. Jak wspomniano powyżej ciąg znaków „10:30” może być rozpoznany jako 1 lub 3 tokeny.

Podsumowując wykorzystane i przetestowane metody tokenizacji dają bardzo dobre wyniki i są wystarczające do większości zastosowań.

## *3.2. Wykrywanie końca zdania*

Poza podziałem tekstu na wyrazy lub też tokeny bardzo często potrzebne jest wydzielenie większych struktur, czyli zdań. Innymi słowy chodzi o wstawienie w odpowiednim miejscu tekstu (lub uporządkowanego zbioru tokenów) znaków końca zdania, w skrócie ZKZ. Podział taki może być potrzebny na dalszych etapach analizy. Przykładowo realizacja rozbioru zdania jest znacznie łatwiejsza, jeśli analizujemy każde zdanie oddzielnie. Również w rozwiązaniach sumaryzacji polegających na wybieraniu najistotniejszych zdań zakłada się, że końce zdań są oznaczone lub łatwe do zidentyfikowania.

Intuicyjne rozwiązanie polega na wykorzystaniu znaków przystankowych i wielkości liter. Wiąże się to jednak z problemem rozwiązywania niejednoznaczności interpretacji znaków przystankowych, w szczególności kropki. Kropka może występować jako znak końca zdania, jako część skrótu lub nazwy własnej lub też może spełniać obie funkcje jednocześnie. Rozwiązaniem tego problemu może być wykorzystanie listy skrótów, a także heurystyki opierającej się na występowaniu wielkich liter. Trudnością w tego typu rozwiązaniach jest przygotowanie listy skrótów, a także poprawne rozpoznawanie funkcji kropki w nazwach własnych. W szczególności dla każdej dziedziny analizowanych tekstów powinna być przygotowana odpowiednia lista skrótów.

Teoretycznie istnieje możliwość wykrywania końca zdania bez użycia znaków przystankowych (np. w tekście, który został ich pozbawiony). Wymaga to jednak bardzo zaawansowanej analizy morfologicznej i syntaktycznej. Zadanie takie nawet dla człowieka sprawnie posługującego się danym językiem naturalnym nie zawsze jest łatwe, a uzyskane wyniki są często niejednoznaczne lub wymagają odwoływania się do szerszego kontekstu lub zewnętrznej wiedzy. Możliwa jest realizacja takiego zadania w sposób maszynowy z wykorzystaniem metod statystycznych i probabilistycznych, np. na podobnej zasadzie jak opisane w rozdziale 3.8.2.

W literaturze zostało zaproponowanych wiele rozwiązań problemu wykrywania końca zdania i radzenia sobie z niejednoznacznością interpretacji kropki. W [Greffenstette, Tapanainen, 1994] zaprezentowana została półautomatyczna adaptacyjna metoda usuwania niejednoznaczności, która wykorzystuje listę skrótów. W [Mikheev, 2000] zostało zaproponowane podejście bazujące na analizatorze morfologicznym trenowanym na nieopisanych danych. W [Schmid, 2000] opisana jest statystyczna metoda usuwania niejednoznaczności interpretacji kropki, która nie wymaga ani opisanych danych trenujących, ani informacji leksykalnych poza korpusem tekstu, który ma zostać podzielony na zdania. Metoda wykorzystująca maksymalną entropię została zaprezentowana w [Reynat, Ratnaparkhi, 1997]. Stosowane są również metody korzystające z sieci neuronowych, czego przykładem jest [Palmer, Hearst, 1994]. W [Piskorski, 2001] wykrywanie końca zdania zostało przeniesione na późniejszy etap analizy (po rozpoznaniu nazw własnych), co pozwala dodatkowo uprościć interpretację kropki do kilku prostych heurystyk.

W ramach prac badawczych zaproponowano i przetestowano rozwiązanie dla tekstów w języki polskim bazujące na liście skrótów, wielkości liter i kilku dodatkowych heurystykach. Z powodów podanych powyżej omówione dalej rozwiązanie korzysta ze znaków interpunkcyjnych i informacji o wielkości liter. Zdanie powinno być zakończone odpowiednim znakiem interpunkcyjnym. Jednak nie zawsze znaki takie oznaczają koniec zdania. W szczególności problemy sprawia występowanie kropki w skrótach, a także wielokropki. Poniżej opisany jest algorytm opracowany w ramach [Proj1]:

- Jedynymi znakami mogącymi kończyć zdanie (ZKZ – znak końca zdania) w języku polskim są [. ! ?]
- Jeśli po ZKZ występuje mała litera, to ZKZ nie kończy zdania (np.: „*...oraz, m. in. mój ojciec...*”)
- Jeśli po kropce występuje duża litera, to sprawdzana jest lista skrótowców (na liście tej znajdują się skróty, po których wystąpieniu może pojawić się wyraz należący do tego samego zdania). Jeśli przed kropką występuje skrótowiec, to program nie kończy zdania – jeśli nie – to program uznaje zdanie za skończone. (np.: „*Pan dyr. Maliniak...*”, „*...kredka, itd. Maliniak poszedł do domu.*”)
- Jeśli po ZKZ występuje znak razem z ZKZ tworzący parę, która pojawia się na liście symboli, to zdanie nie jest kończone (przykłady poniżej)
- Jeśli po kropce występuje cyfra, to sprawdzane jest na odpowiedniej liście, czy przed kropką nie znajduje się wyraz, po którym cyfra może wystąpić (np. „*godz. 13*”). Jeśli tak – zdanie nie jest kończone. Jeśli nie, to sprawdzane jest, czy kropkę otaczają bezpośrednio cyfry (tzn. bez znaków białych pomiędzy nimi, np.: „*13.45*”) – jeśli tak, to mamy najprawdopodobniej do czynienia z godziną lub datą i zdanie nie jest kończone – w przeciwnym przypadku – jest (np.:”*...było ich 124. 121 poszło...*”).
- Wielokropek kończy zdanie, dwie po sobie następujące kropki – nie (gdyż zwykle są używane do zaznaczania wycięcia fragmentu tekstu, często wewnątrz jednego zdania („to [..] jest nie do pomyślenia!”)

- również konstrukcje typu „!?!” są traktowane spójnie i rozpatrywane wg zasad dla ostatniego znaku, gdyż zwykle wyrażają emocje i mogą stać w środku zdania
- w pozostałych przypadkach, a w szczególności, jeśli za ZKZ pojawi się inny znak – zdanie jest kończone.

Algorytm korzysta z trzech słowników:

- Lista skrótowców zakończonych kropką, po których może wystąpić wyraz należący do tego samego zdania (np.: „np.”, dla porównania – po „itd.” – nie może) – tu możliwe jest używanie skrótowców „podwójnych”, takich jak „m. in.”.
- Lista skrótowców zakończonych kropką, po których może wystąpić cyfra należąca do tego samego zdania (np.: „godz.”)
- Lista par symboli, z których pierwszy jest znakiem kończącym zdanie. Wystąpienie takiej pary w tekście nie powoduje zakończenia zdania, np. para znaków: „.)”.
  Przykład zdania: *„Wszystko było widać (Choć jeszcze nikt tego nie udowodnił.) jak na dłoni.”*

Słowniki takie powinny być uzupełniane i modyfikowane w zależności od dziedziny, gdyż np. skrótowce mogą mieć w różnych środowiskach różne znacznie i różną łączność z innymi wyrazami.
Powyższy algorytm zaimplementowano w [Proj1] i testowano na różnorodnych przykładach mających na celu wykazanie jego słabych stron.

Jego największym minusem jest całkowite nie radzenie sobie ze skrótami nazw własnych, a w szczególności imion i nazwisk (oczywiście można dodać jednoliterowe skróty do pliku abbrev.txt, ale nie rozwiąże to problemu, choć może trochę usprawnić działanie programu na szczególnych klasach tekstów). Sam problem jest jednak bardzo złożony, co uwidacznia chociażby poniższy przykład:

*Ludzie nie chcieli już słyszeć o środku K. R. Marciniak wstał i podszedł do biurka.*

Poprawne podzielenie tego tekstu na dwa zdania jest możliwe tylko wtedy, gdy czytelnik zanurzony jest w kontekście i wie, czy ów środek, o którym mowa może mieć

kryptonim K. R. albo K. Pomocna może się okazać także wiedza, co do imienia Pana Marciniaka.

Algorytm może sprawiać również drobne problemy w przypadku cytatów obejmujących swoim zakresem całe zdanie, gdyż wbudowane reguły i słownik symboli pozwalają bądź traktować znaki cudzysłowów jak inne znaki bądź nie kończyć zdania zawsze, gdy napotka na cudzysłów za kropką. Druga metoda wydaje się dawać lepsze rezultaty, gdyż często zdania cytowane są częścią „nadzdania":

*Powiedział: „To jest pole minowe?" i ruszył naprzód.*

Metodę zagłębiającą („nawiasową") odrzucono, gdyż jeden niewłaściwie postawiony symbol nawiasu, bądź cudzysłowu mógłby zaburzyć sporą część procesu wyszukiwania zdań, a poza tym symbole te są używane również w innych miejscach, a nie tylko do zanurzania treści (nawias – np. do list numerowanych, cudzysłów – np. do oznaczania sekund kątowych).

Pomimo przedstawionych powyżej ograniczeń, algorytm dobrze radzi sobie z dzieleniem tekstu na zdania, a lokalność działania ograniczająca się do co najwyżej jednego symbolu niebiałego naprzód i dwóch słów wstecz od ZKZ powoduje, że miejsca podziału tekstu na zdania są w miarę niezależne, co skutkuje wyeliminowaniem wpływu błędnego rozpoznania na kolejne decyzje.

### 3.2.1. Rozpoznawanie kontekstu zdania

Poza podziałem tekstu na zdania ważnym i potencjalnie przydatnym zadaniem jest rozpoznawanie kontekstu zdania. Podstawowym celem tego zadania jest określenie czy zdanie jest częścią dialogu, listy, wyliczenia, tabeli czy też występuje jako część tekstu ciągłego. Informacja taka może być przydatna na dalszych etapach przetwarzania. Przykładowo w procesie klasyfikacji tekstów lub tworzenia streszczeń można przypisywać różną ważność zdaniom występującym w dialogach i poza nimi.

W ramach opracowanej platformy zdefiniowano odpowiednie rodzaje kontekstu. Format ich opisu znajduje się w części Załącznik A: Środowisko implementacyjne (str. 134).

W [Proj2] opracowano i zaimplementowano algorytm wykrywania końca zdania oraz rozpoznawania kontekstu zdania. W procesie wykrywania końca zadania

wykorzystywane są podobne metody jak w [Proj1]. Jednak projekt ten dodatkowo rozszerzono o wykorzystywanie informacji o układzie tekstu, np. pustych linii, wyliczeń itp. Zakończenie zdania wykrywane jest w sytuacji, gdy pobrano token typu „konieclinii" oraz:

1. ostatnim tokenem aktualnego zdania jest token typu „znak" i tym znakiem jest jeden ze znaków: .:, :, " – w ten sposób kończą się zdania wprowadzające np.: listy
2. ostatnim tokenem aktualnego zdania jest token typu „konieclinii" – jest to sytuacja, gdy wystąpi jedna linia pusta pomiędzy zdaniami
3. następnym tokenem jest „•" (ang. ***bullet***) lub ciąg cyfr, a następnie opcjonalnie występuje kropka i nawias okrągły zamykający, a jednocześnie nie jest to token typu: data, interwał daty, symbol, skrót, liczba rzeczywista. Odpowiada to sytuacji, gdy jedno zdanie się kończy, a drugie rozpoczyna się od listy, np.:

   *Lista:*

   • *zdanie 1 z listy*

   • *zdanie 2 z listy*

4. następnym tokenem jest token typu tabulator, a kolejnym tokenem nie jest token typu tabulator – warunek ten dodano w trakcie optymalizacji reguł kończenia zdania do przypadków testowych – nie ma on jednoznacznego odniesienia do reguł języka polskiego.

Po podzieleniu tekstu na zdania wykonywane jest rozpoznanie kontekstu w oparciu o następujące reguły:

a) kontekst „nagłówek" ustawiany jest gdy zdanie kończy się znakiem ':' oraz przed jak i po zdaniu są puste linie
b) kontekst „dialog" ustawiany jest gdy:
   i. poprzednie zdanie zakończyło się znakiem końca linii, a bieżące zdanie zaczyna się od myślnika
   ii. zdanie zaczyna się od sekwencji: koniec linii, myślnik

iii. gdy poprzednie zdanie miało kontekst dialog, to dialog jest kontynuowany tak długo, aż wystąpi znak końca linii.

c) kontekst „lista" ustawiany jest gdy:
   i. poprzednie zdanie kończy się sekwencją: ':' i znakiem końca linii
   ii. poprzednie zdanie ma kontekst lista, a aktualne zdanie zaczyna się od jednej z kombinacji: tabulator, liczba-kropka, liczba-nawias, myślnik

d) w pozostałych przypadkach ustawiany jest kontekst „zwykły"

Przeprowadzono testy na wybranych dokumentach testowych. Wyniki testu przedstawia poniższa tabela.

| **Plik** | **Liczba wykrytych zdań** | **Liczba zdań poprawnie zakwalifikowanych** | **Poprawność** |
|---|---|---|---|
| Architekci | 69 | 66 | 95% |
| Bankowa superfuzja… | 105 | 102 | 97% |
| Chińska nauka… | 93 | 93 | 100% |
| lalka1 | 125 | 123 | 98% |
| MAGNETYZER | 62 | 55 | 89% |
| Mechanizm różnicowy | 17 | 16 | 94% |
| Młodzi w Polsce… | 35 | 32 | 91% |
| naglowki | 134 | 111 | 83% |
| Pierwsze dziecko… | 49 | 47 | 96% |
| Seleukos | 131 | 126 | 96% |
| we_test | 23 | 21 | 91% |

Jak widać zastosowane podejście charakteryzuje się bardzo wysoką skutecznością. Kluczowym elementem jest tutaj analiza kontekstowa, czyli uwzględnianie typu zdania poprzedzającego. Dzięki temu uzyskano wysoką skuteczność wykrywania dialogów,

które występują w dużej ilości w plikach *„lalka1”* oraz *„Chinska nauka…”*. Najgorsze wyniki uzyskano w przypadku pliku *„naglowki”*. Jest to dokument zawierający m.in. nagłówki pocztowe SMTP oraz fragmenty skryptów systemu UNIX, co utrudnia analizę tekstu i wykracza poza zakres analizy języka polskiego.

Uzyskane wyniki są bardzo dobre. Dodatkowo dzięki uniwersalności i konfigurowalności stworzonych narzędzi istnieje możliwość dopasowania do dziedziny, np. uzupełnianie słowników skrótów itp.

## 3.3. Analiza morfologiczna

Uporządkowany zbiór tokenów poddać należy analizie morfologicznej. Składa się ona z dwóch powiązanych ze sobą zadań:

- znajdywania form podstawowych wyrazów (**lematów**), czyli tzw. ***stemming***'u,
- rozpoznawania i oznaczania części mowy i ich form.

W wielu zadaniach TDM zastosowanie stemming'u pozwala na wyraźną poprawę rezultatów. Ma to szczególne znaczenie, jeśli porównujemy słowa lub zbiory słów (dokumenty, klasy dokumentów), np. przy klasyfikacji lub grupowaniu. Porównajmy dwa zdania:

*Zawodnik startuje w wyścigu.*

*Zawodnicy startowali w wyścigach.*

Pomimo wyraźnego podobieństwa znaczeniowego, a na pewno tematycznego, bezpośrednie porównanie wyrazów może doprowadzić do znalezienia wyłącznie jednego zgodnego wyrazu, a mianowicie przyimka „w”. Dodatkowo w wielu sytuacjach słowo to może, jako bardzo często występujące, być wykluczone z analizy (patrz rozdział 2.2 „Metody przekształcania reprezentacji”). W takim wypadku oba zdania będą uznane za zupełnie niezgodne. Jeśli jednak sprowadzimy wszystkie wyrazy do formy podstawowej, to w wypadku obu zdań uzyskamy ten sam wynik:

*Zawodnik startować w wyścig.*

Powyższy przykład obrazuje przydatność takiego podejścia. Stąd od dawna trwały próby rozwiązania problemu skutecznej realizacji stemming'u. Odmiana wyrazów w języku angielskim występuje w ograniczonym zakresie. Pełne rozwiązanie dla tego

języka zostało podane w [Porter, 1980]. W językach silnie fleksyjnych, np. polskim, problem jest bardziej złożony. W niektórych zastosowaniach wystarczają rozwiązania heurystyczne. Przykładem takiego rozwiązania jest obcinanie końcówek wyrazów. Opiera się ono na obserwacji, że fleksja dotyczy zwykle końcówek, a znacznie rzadziej wpływa na rdzeń wyrazu. Korzystając ze zdań z poprzedniego przykładu można skrócić wyrazy uzyskując:

*Zawodnik startuje w wyścigu.*

*Zawodnicy startowali w wyścigach.*

W ten sposób również uzyskuje się podobny cel. Trudno jest jednak znaleźć prostą metodę określania ile liter wyrazu ma być pozostawianych, a ile usuwanych. Oczywiście można przyjąć arbitralnie jakąś liczbę. W pewnych wypadkach spowoduje to jednak uzyskanie błędnych wyników, np.

*wybielać - wybierać*

*trawa - trawienie*

Taka metoda stemming'u jest również nieskuteczna w wypadku jakichkolwiek nieregularności odmiany. Dodatkowo nie uzyskujemy poprawnych form podstawowych, a jedynie ich aproksymację. Z tego powodu w wielu wypadkach taka metoda jest niewystarczająca.

Podejmowane próby wskazują, że w celu realizacji pełnego stemming'u w językach fleksyjnych, w szczególności w polskim, konieczne jest wykorzystanie słownika oraz zestawu reguł odmiany. Tego typu rozwiązanie ma dwie zalety:

- uzyskiwana forma podstawowa jest w pełni poprawną formą wyrazu,
- reguły pozwalają na uzyskanie informacji o rodzaju części mowy oraz rodzaju formy fleksyjnej.

Tokeny zawierające pełną informację o formach podstawowych, rodzaju części mowy i ich formach są podstawą dalszej analizy STP. Rozszerzona informacja o wyrazach pozwala również na implementację rozwiązań TDM opierających się na formach wyrazów. Można sobie przykładowo wyobrazić klasyfikację tekstów nie na podstawie

słownictwa, ale na podstawie używanych form wyrazów, co pozwala odróżnić teksty formalne od nieformalnych.

Oprócz analizy morfologicznej ciekawym aspektem jest **synteza morfologiczna**, czyli generowanie odpowiedniej formy części mowy na podstawie formy podstawowej i atrybutów opisujących formę. Dysponując możliwością analizy i syntezy można przeprowadzać interesujące transformacje tekstu. Np. podczas pisania oferty na wdrożenie systemu informatycznego do opisu funkcjonalności często używany jest czas przyszły. Następnie po wdrożeniu duża część tego opisu jest kopiowana do dokumentacji, lecz musi być zmieniona na czas teraźniejszy. W tym celu wystarczy przeprowadzić analizę morfologiczną, następnie w uzyskanych tokenach zmienić atrybut „czas" na „teraźniejszy", a następnie przeprowadzić syntezę morfologiczną. Warto zauważyć, że nie jest wówczas potrzebna głębsza analiza, np. rozbiór zdań.

### 3.3.1. Rozpoznawanie poszczególnych części mowy

W ramach prac badawczych prowadzonych pod kierownictwem autora niniejszej rozprawy powstało kilka implementacji zarówno stemmera [Proj4], jak i analizatora morfologicznego m.in. [Proj5], [Proj6]. Wszystkie implementacje wykorzystywały polską wersję słownika ISpell (pliki reguly.txt oraz slownik.txt) rozwijanego przez Piotra Gackiewicza oraz Włodzimierza Macewicza.

Słownik ten został stworzony w celu sprawdzania pisowni w plikach tekstowych, która wymaga od niego tworzenia form pochodnych z form podstawowych, co pociągnęło za sobą stworzenie specyficznych struktur danych (pozwalają one na łatwe przypisanie pewnego typu odmiany do danego wyrazu, a następnie na podstawie załączonych reguł wygenerowanie formy pochodnej celem sprawdzenia czy pokrywa się ona z podanym wyrazem). Analizator morfologiczny przeprowadza proces odwrotny – na podstawie danej formy pochodnej znalezienie formy podstawowej oraz określenie atrybutów formy pochodnej. Dlatego w niniejszej pracy opracowano algorytmy realizacji takiego przetwarzania. Słownik ten rozszerzono o odpowiednie flagi pozwalające rozpoznać część mowy i jej formę.

Oprócz analizy morfologicznej zaimplementowano synteza morfologiczna.

Potencjalnie wyraźnej poprawy wyników analizy morfologicznej można się spodziewać w przypadku wykorzystania korpusu języka polskiego, który jest przygotowywany w Polskiej Akademii Nauk [Przepiórkowski, 2004], [Przepiórkowski, 2005], [Przepiórkowski,2, 2005]. Zawiera on bardzo rozbudowany zbiór dokumentów w języku polskim, które zostały anotowane (opisane formami morfosyntaktycznymi). Korpus ten pozwoli na uzyskanie bardzo wysokiej poprawności analizy morfologicznej oraz usuwania niejednoznaczności.

Publicznie dostępnym narzędziem do przeprowadzania analizy morfologicznej w języku polskim jest system Morfeusz [Morfeusz] autorstwa Marcina Wolińskiego, do którego dane przygotował Zygmunt Saloni. Na potrzeby niniejszej rozprawy analizatory morfologiczne przygotowano jednak niezależnie, co miało dodatkowy walor edukacyjny.

**Przysłówki wtórne**

1. Sprawdź czy końcówką wyrazu jest „ej” (wszystkie przysłówki wtórne stopnia wyższego i najwyższego mają właśnie tę końcówkę). Jeśli tak – przejdź do punktu 2, w przeciwnym wypadku umieść wyraz na liście pomocniczej i przejdź do punktu 5.
2. Sprawdź czy słowo wejściowe nie jest przymiotnikiem rodzaju żeńskiego w dopełniaczu za pomocą reguł flagi „x” (ta forma również ma końcówkę „ej”). Jeśli tak - zakończ algorytm, w przeciwnym wypadku przejdź do punktu 3.
3. Umieść lemat (wyraz po obcięciu końcówki „ej”) na liście pomocniczej. Sprawdź oboczności lematu (wymiana zębowych twardych na palatalne, wymiana welarnych twardych na funkcjonalnie miękkie, wymiana sonornych, wymiana wargowych na grupy dwufonemowe, wymiana funkcjonalnie miękkich na twarde), czyli zamień końcówki lematu według następujących reguł: „ci” na „t” lub „k”; „dzi” na „d” lub „dz”; „si” na „s”; „zi” na „z”; „ni” na n”; „wi” na „w”; „bi” na „b”; „mi” na „m”; „ści” na „st”; „ździ” na „zd”; „ż” na „g”; „rz” na „r”; „l” na „ł”; „c” na „t”. Dla każdej powstałej formy sprawdź, czy rozpoczyna się od „naj”. Jeśli tak to usuń prefiks i dodaj do wyrazu flagę wskazującą na stopień najwyższy. W przeciwnym wypadku dodaj flagę stopnia wyższego. Następnie umieść tak powstałe formy na liście pomocniczej.

4. Do każdej formy z listy pomocniczej dodaj końcówkę „o”, „e”, „ko”, „eko” lub „oko”, tworząc w ten sposób możliwe formy przysłówka stopnia podstawowego.
5. Dla każdego wyrazu znajdującego się na liście pomocniczej sprawdź na podstawie reguł słownikowych, czy można utworzyć wyraz od przymiotnika (flaga y odpowiadająca za tworzenie przysłówków w stopniu podstawowym z przymiotnika). Jeśli wyrażenie regularne ze słownika reguł pasuje, przypisz wyrazowi część mowy *przysłówek* z odpowiednią flagą (brak przypisanej flagi oznacza stopień podstawowy).

#### Liczebniki

Algorytm korzysta ze słownika oraz zbioru reguł.

1. Sprawdź czy wyraz wejściowy na podstawie reguł zawartych w słowniku (flag i odmian) jest liczebnikiem. Jeśli tak to zakończ algorytm, w przeciwnym wypadku zapisz formy podstawowe na liście i przejdź do punktu drugiego.
2. Dla każdego wyrazu na liście sprawdź czy kończy się na „krotny”. Jeśli tak to jest to liczebnik wielokrotny. Sprawdź czy wyraz zawiera ciągi „wielo”, „kilku”, „kilko”, jeśli tak to jest to liczebnik nieokreślony, w przeciwnym wypadku jest to liczebnik określony. Zakończ algorytm.
3. Dla każdego wyrazu na liście sprawdź czy kończy się na „tysięczny”, „milionowy” lub „miliardowy”. Sprawdź czy wyraz nie zawiera ciągów „wielo”, „kilku”, „kilko” (gdyż są to raczej przymiotniki, zbiór liczebników porządkowych, nieokreślonych został wpisany do słownika, a więc zostaną rozpoznane w punkcie 1 algorytmu). Spełnienie powyższych warunków określa część mowy - liczebnik porządkowy, określony. Zakończ algorytm.
4. Dla każdego wyrazu na liście sprawdź, czy kończy się na „raki”. Jeśli tak i wyraz nie zawiera ciągu „różno” to jest to liczebnik wieloraki. Sprawdź czy wyraz zawiera ciągi „wielo”, „kilku”, „kilko” – jeśli tak, to jest to liczebnik nieokreślony, w przeciwnym wypadku - określony. Zakończ algorytm.

### Zaimki

Przy rozpoznawaniu zaimków wykorzystywany jest plik dodatkowy słownik, który zawiera pary wyrazów: forma odmieniona i podstawowa oraz flagi określające przypadek, liczbę, rodzaj, osobę.

Zawartość pliku wczytana jest do struktury mapy. Kluczem jest wyraz odmieniony, natomiast wartością - lista części mowy, które odpowiadają wyrazowi.

Algorytm: Sprawdź czy kluczem w mapie jest wyraz wejściowy. Jeśli tak to zwróć listę możliwych części mowy (wartość klucza), w przeciwnym razie zakończ algorytm.

### Przyimki, przysłówki pierwotne

Sprawdź czy kluczem w mapie jest wyraz wejściowy. Jeśli tak to zwróć listę możliwych części mowy (wartość klucza), w przeciwnym razie zakończ algorytm.

### Rzeczowniki, przymiotniki, czasowniki

Algorytm korzysta ze słownika oraz zbioru reguł.

Najpierw sprawdzane jest, czy dane słowo jest rozpoznawane za pomocą zdefiniowanych reguł. Jeśli nie, kolejno odcinane są litery z końca słowa i wyszukiwane jako końcówki w bazie reguł przetwarzania. (Wcześniej w podobny sposób sprawdzane są prefiksy). Jeśli reguła zostanie znaleziona, to jest stosowana (wraz z wyrażeniem regularnym stosowalności iSpella). Uzyskane słowo jest sprawdzane w słowniku, aby określić, czy dla takiej postaci można stosować daną regułę. Jeśli jest włączony tryb heurystyk, to słowo takie nie musi się znaleźć w słowniku, pod warunkiem, że reguła jest tak często używana (wówczas może być używana jako heurystyka).

### Nakaźniki, wykrzykniki, spójniki, modulanty

Ponieważ są to nieodmienne części mowy, wyrazy należące do tej kategorii są *explicite* umieszczone w odpowiednim słowniku razem z odpowiednimi flagami. Za pomocą tychże flag możliwe jest jednoznaczne scharakteryzowanie kategorii danego słowa.

### 3.3.2. Ogólna koncepcja algorytmu analizy

1. Pierwszy krok takiej analizy to sprawdzenie czy wyraz jest wyjątkiem. Jeśli tak to dalsza analiza jest zbędna. Atrybuty wyrazu pobierane są ze słownika wyjątków.
2. Drugi krok to sprawdzenie czy wyraz występuje w słowniku. Jeśli tak to, jest to wyraz w formie podstawowej. Wyraz jest więc rozpoznany. Atrybuty ustawiane są na podstawie flag ze słownika. Dalsza analiza nie jest przeprowadzana.
3. Trzeci krok to sprawdzanie czy badany wyraz pasuje do którejś z reguł. Wyraz może pasować do wielu reguł. Wtedy jako wynik zwracane są wszystkie możliwości.
4. Jeśli w trzecim kroku nie udało się sklasyfikować wyrazu, to stosowane są heurystyki. Heurystyki działają wtedy, gdy dla danego słowa została znaleziona pasująca reguła lub zbiór reguł, jednak po jej zastosowaniu zostało znalezione w słowniku słowo podstawowe. Heurystyka wybiera jedną z reguł i na jej podstawie tworzy formę podstawową oraz identyfikuje część mowy oraz atrybuty ( przypadek, rodzaj ... ). Oczywiście nie zawsze heurystyka daje poprawną odpowiedź.

### 3.3.3. Ogólna koncepcja algorytmu syntezy

W celu wyznaczenia nowej formy słowa wykonywane są następujące kroki:

1. W pierwszym kroku sprawdzane jest czy badany wyraz nie jest wyjątkiem. W strukturze wyjątków wyszukiwane są te wyrazy, których forma podstawowa zgadza się z formą podstawową odczytaną z tokena. Jeśli tak to oznacza, że został znaleziony właściwy wyraz. Algorytm kończy działanie.
2. Jeśli powyższy krok nie dał efektu to w drugim kroku sprawdzane jest czy wyraz jest formą podstawową. Jeśli jest to wyraz w mianowniku liczby pojedynczej lub też jest to bezokolicznik to algorytm kończy działanie.
3. Jeśli zawiodły poprzednie kroki, to w trzecim kroku poszukiwana jest pasująca reguła. Wyszukiwane są te reguły, w których końcówka zamieniana równa jest końcówce odczytanej formy podstawowej. Odczytywane są też flagi ze

słownika zapisane przy formie podstawowej. Sprawdzane jest następnie czy flaga ze znalezionej reguły występuje wśród flag odczytanych ze słownika oraz czy atrybuty słowa odczytane z tokena zgadzają się z tymi odczytanymi z reguły. Jeśli tak to stosowana jest znaleziona reguła. Odcinana jest końcówka od formy podstawowej, a dodawana jest końcówka formy pochodnej.

### 3.3.4. Szczegółowy opis stosowania reguł

- Zastosowanie reguł przyrostkowych: Odcinane są od wyrazu końcówki o coraz większej długości. Następnie szukane są reguły, w których taka końcówka jest obcinana. Jeśli taka reguła zostanie znaleziona, to tworzona jest na jej podstawie forma podstawowa wyrazu i sprawdzane jest czy znajduje się ona w słowniku. Jeśli tak to na podstawie flag ze słownika oraz flag ze znalezionej reguły określane są wszystkie atrybuty wyrazu.
- Jeśli forma podstawowa nie została znaleziona w słowniku wtedy ponownie poszukiwana jest pasująca reguła. Lecz tym razem dla formy podstawowej uzyskanej z pierwszej reguły. Jeśli to poszukiwanie nie zakończy się sukcesem (nie zostanie znaleziona reguła lub też dla znalezionej reguły zostanie znaleziona w słowniku forma podstawowa), to przerywany jest ten krok analizy. Jeśli natomiast zostanie znaleziona reguła pasująca do formy podstawowej a uzyskana z niej wtórna forma podstawowa występuje w słowniku, to możemy mieć do czynienia z jednym z następujących przypadków, w których badany wyraz to:
    - przysłówek odimiesłowowy ( *śpiewać >> śpiewający >> śpiewająco*)
    - imiesłów czynny odimiesłowowy (*czytać >> czytający >> czytając*)
    - odmiana imiesłowu (*czytać >> czytający >> czytającego*)
- W tych szczególnych przypadkach wyraz należący do słownika otrzymywany jest dopiero po dwukrotnym zastosowaniu reguł. Ostatecznie atrybuty dla badanej części mowy odczytywane są na podstawie flag obu użytych reguł oraz flag znalezionego w słowniku słowa podstawowego.

- Sprawdzenie czy wyraz jest stopniem najwyższym przymiotnika lub przysłówka. Najpierw sprawdzane jest czy zaczyna on się od „naj”. Jeśli tak to odcinany jest ten przedrostek i sprawdzane jest czy otrzymane słowo występuje w słowniku jako forma wyższa przymiotnika, jeśli nie to poszukiwana jest odpowiednia reguła.
- Zastosowanie reguł przedrostkowych: Sprawdzenie czy wyraz nie rozpoczyna się od jednego z przedrostków ( np. anty-, nie-, nad-, itp.). Jeśli tak to odcinany jest przedrostek, po czym otrzymany wyraz sprawdzany jest według poprzednich punktów.

Powyższe kroki pozwalają wykryć nawet dosyć złożone wyrazy np. nienajlepszego.

*lepszy >> najlepszy >> nienajlepszy >> nienajlepszego*

### 3.3.5. Działanie heurystyk

Heurystyka jest stosowana, gdy reguły nie dały rozwiązania. Wyszukiwane są wszystkie reguły pasujące do danego wyrazu (za wyjątkiem reguł, w których nie dodawana jest żadna końcówka). Następnie wybierana jest jedna z nich. Wybierana jest ta reguła, która dotychczas była najczęściej używana. Takie podejście daje duże prawdopodobieństwo na to, że użyta zostanie właściwa reguła. Im więcej słów już zostało przebadanych tym lepiej określona jest częstotliwość użycia poszczególnych reguł i tym skuteczniejsze są heurystyki. Oczywiście sposób powyższy nie gwarantuje nam, że zostanie uzyskany poprawny wynik. Pozwala jednak prawidłowo rozpoznać wyrazy nie występujące jeszcze w bazie. Np. słowo „uniksowego” zgodnie z tą metodą zostanie odczytane jako przymiotnik o formie podstawowej „uniksowy”.

#### Testy

W ramach [Proj5] zaprojektowano i zrealizowano moduł morfologiczny w środowisku Java. Pozwala on na przeprowadzenie analizy i syntezy morfologicznej dokumentów w języku polskim. W celu sprawdzenia jego skuteczności przeprowadzono testy na wybranych plikach testowych. Po przeprowadzeniu analizy morfologicznej wyniki sprawdzono manualnie. Zostały policzone tokeny:

- A: rozpoznane całkowicie poprawnie
- B: rozpoznane częściowo poprawnie – rozpoznana część mowy, nie wszystkie właściwości rozpoznane poprawnie
- C: nierozpoznane

Na tej podstawie zostały określone miary poprawności.

Poprawność całkowita:

$$PC = A / (A+B+C)$$

Poprawność rozpoznania części mowy:

$$PRCM = (A+B) / (A+B+C)$$

Korzystając z wyników i doświadczeń [Proj5] został zrealizowany [Proj6]. Miał on na celu poprawę uzyskanych wyników poprzez uzupełnienie zbioru reguł słownikowych oraz optymalizację wydajnościową zastosowanych konstrukcji programistycznych. W tym projekcie implementacja opierała się o platformę .NET. Decyzja ta była podyktowana chęcią sprawdzenia w praktyce niezależności stworzonych rozwiązań od platformy programistycznej i wykrycia ewentualnych ograniczeń uniwersalności rozwiązania. Dodatkowym argumentem była możliwość porównania wydajności obu platform programistycznych. Wykonaną implementację przetestowano na tym samym zbiorze plików zgodnie z przedstawioną powyżej metodologią.

Wyniki porównania przedstawia poniższa tabela.

| Plik | Liczba tokenów | [Proj5] | | [Proj6] | |
|---|---|---|---|---|---|
| | | PC | PRCM | PC | PRCM |
| Architekci.txt | 872 | 65 % | 80 % | 79% | 90% |
| lalka1.txt | 1675 | 64 % | 81 % | 73% | 85% |
| MAGNETYZER.txt | 948 | 67 % | 81 % | 79% | 89% |
| Mechanizm roznicowy.txt | 387 | 66 % | 80 % | 67% | 79% |
| we_test.txt | 208 | 55 % | 71 % | 69% | 85% |

Jak widać w powyższej tabeli poprawność całkowita uzyskana w [Proj5] znajdowała się w przedziale 55%-67%. W [Proj6] wyniki uległy poprawie i znajdują się w przedziale 67%-79%. Również poprawność rozpoznawania części mowy uległa poprawie z przedziału 71%-81% w [Proj5] do 79%-90% w [Proj6].

Uzyskane wyniki wymagają szerszego omówienia. Generalnie uzyskana poprawność jest dosyć dobra i większość tokenów opisywana jest w pełni poprawnie. Jednocześnie można wnioskować, że uzupełnianie słownika i zestawu reguł może prowadzić do dalszej poprawy wyników. Jednak poprawność na poziomie nawet 80%-90% oznacza, że statystycznie jeden token na 5-10 tokenów jest błędnie opisany. Oznacza to, że prawie w każdym zdaniu pojawi się błąd w opisie tokena. Niektóre błędy mają drugorzędne znaczenie, lecz niektóre z nich mogą generować błędy na dalszych etapach przetwarzania. Przykładowo niewłaściwie rozpoznana część mowy może wręcz uniemożliwić przeprowadzenie rozbioru zdania. Można stąd wnioskować, że o ile przetestowana metoda analizy morfologicznej daje dobre rezultaty, to jednak konieczne jest uzupełnienie słownika i zbioru reguł, aby znacząco zredukować poziom błędów.

Oprócz poprawności wyników porównano również czasy przetwarzania w przypadku zastosowania obu rozwiązań, gdyż jednym z założeń [Proj6] była optymalizacja wydajnościowa. Poniżej przedstawiono czasy wykonania dla wybranych plików testowych. Test przeprowadzono na komputerze z procesorem Athlon XP 1,8 GHz oraz 512 MB RAM.

| **Plik** | **Czas wykonania [s]** | | **Przyrost wydajności [krotność]** |
|---|---|---|---|
| | **[Proj5]** | **[Proj6]** | |
| architekci.xml | 0,688 | 0,094 | 7,3 |
| lalka1.xml | 1,063 | 0,140 | 7,6 |
| magnetyzer.xml | 0,766 | 0,078 | 9,8 |
| mechanizm roznicowy.xml | 0,469 | 0,047 | 10,0 |
| nagłówki.xml | 1,359 | 0,250 | 5,4 |
| seleukos.xml | 1,156 | 0,125 | 9,2 |
| we_test.xml | 0,297 | 0,030 | 9,9 |
| Mlodzi.xml | 0,406 | 0,047 | 8,6 |
| Pierwsze dziecko z przeszczepu jajnika innej kobiety.xml | 0,500 | 0,047 | 10,6 |
| Opole 2005 - Tylem do sceny .xml | 0,656 | 0,063 | 10,4 |
| Bankowa superfuzja - UniCredito przejmuje HVB.xml | 0,797 | 0,078 | 10,2 |

Jak widać w przedstawionej powyżej tabeli nastąpił 5 do 10-krotny przyrost wydajności. Przyczyną tego faktu jest głównie zastosowanie wydajniejszych struktur danych i konstrukcji programistycznych. Istotny wpływ ma również niski poziom zoptymalizowania kodu w [Proj5], gdyż celem tego projektu było przede wszystkim uzyskanie poprawnych wyników, a czasy wykonania nie były optymalizowane, gdyż i tak były stosunkowo krótkie. Różnice wydajności między platformą .NET i Java mogą mieć również pewien udział w uzyskanych różnicach wyników. Jednak według niezależnych testów dostępnych w literaturze można wnioskować, że wpływ ten jest znikomy.

## 3.4. Usuwanie niejednoznaczności

Na poziomie analizy morfologicznej w wielu wypadkach nie jest możliwe jednoznaczne rozpoznanie części mowy. Np. słowo *lecz*, może być zarówno spójnikiem, jak i trybem rozkazującym czasownika *leczyć*. W takich sytuacjach analizator morfologiczny może wygenerować na wyjściu wszystkie możliwe formy. Jeszcze więcej przypadków niejednoznaczności występuje w językach fleksyjnych. Wynika to z faktu, że pomimo występującej odmiany wyrazów niektóre formy są identyczne lub też w niektórych sytuacjach odmiana nie występuje. Przykładowo w języku polskim czasownik w trzeciej osobie liczby pojedynczej w czasie teraźniejszym ma formę niezależną od rodzaju:

*on robi*
*ona robi*
*ono robi*

Usuwanie takiego rodzaju niejednoznaczności nazywane bywa **filtrowaniem części mowy** (ang. ***part-of-speech filtering***) lub **znakowaniem części mowy** (ang. ***part-of-speech tagging***) [Brill, 92]. Usuwanie niejednoznaczności może mieć istotne znaczenie dla dalszych etapów analizy STP, w szczególności rozbioru zdań. Niektóre metody rozbioru zdań radzą sobie z niejednoznacznością i jej usuwanie może być wykonywane dopiero na tym etapie.

Zwykle używane są dwie metody usuwania niejednoznaczności:

1. Wykorzystanie reguł tworzonych przez ekspertów, które wykluczają pewne części mowy w określonym kontekście lub przy spełnieniu jakichś warunków. Reguły mogą też umożliwiać określenie właściwej formy, np. poprzez zachowanie zgodności z wyrazem poprzednim lub następnym. Dużym ograniczeniem tego podejścia jest trudność zdefiniowania zbioru reguł pokrywających wszystkie możliwe sytuacje występujące w języku naturalnym.
2. Zastosowanie podejścia statystycznego, czyli obliczania prawdopodobieństwa występowania danej części mowy i jej formy w kontekście danego zdania. Do tego typu rozwiązań stosowany jest model matematyczny **ukrytych modeli Markova** (ang. ***Hidden Markow Models, HMM***), które są przykładem procesu stochastycznego [Brants, 2000]. Prawdopodobieństwo zwykle jest obliczane poprzez mnożenie prawdopodobieństwa dla krótkich fragmentów zdania, np. 3 wyrazów. Wartości lokalne prawdopodobieństwa mogą być obliczane poprzez analizę dużych korpusów tekstowych z anotowanymi informacjami o częściach mowy i ich formach. Przykłady tego typu podejścia można znaleźć w [Church, 1988] i [Kupiec, 1992]. Proponowane były również podejścia z grupy algorytmów uczących się bez nadzoru, oparte o klasteryzację lub minimalizację wariancji kontekstu, które nie wymagają przygotowywania anotowanych zbiorów trenujących [Pedersen, Bruce, 1997].

### Implementacja

W ramach prac badawczych [Proj8] przeprowadzono eksperymenty z drugim podejściem. Podejście pierwsze nie było sprawdzane eksperymentalnie. Decyzja taka wynikała w dużym stopniu z trudności uzyskania odpowiedniego zbioru reguł eksperckich pokrywającego możliwie duży zakres występujących niejednoznaczności. W podejściu statystycznym reguły te są tworzone automatycznie w procesie uczenia algorytmu. Trudnością jest oczywiście zdobycie odpowiednio dużego zbioru trenującego. Na potrzeby eksperymentu manualnie usunięto niejednoznaczności z wybranego dokumentu. Został on następnie wykorzystany do budowy modelu. W celu określenia wpływu obszerności zbioru trenującego na wyniki przygotowano dwa zbiory o różnej obszerności. Poniżej znajduje się pełniejszy opis implementacji oraz wyników testu.

Implementacja [Proj8] opiera się o model Markowa. Tokenom, dla których znalazła się więcej niż jedna część mowy (forma gramatyczna) przypisywana jest ta z proponowanych części mowy, która zgodnie z modelem była najbardziej prawdopodobna w danym kontekście. Przez kontekst rozumie się części mowy sąsiednich tokenów. Metoda bazuje na modelu, który jest konstruowany w oparciu o duży (oznakowany częściami mowy) tekst. Im większy i „gramatycznie przekrojowy" jest tekst, tym lepszy model może być utworzony.

Zaimplementowane podejście opiera się na spostrzeżeniu, że nawet w tak fleksyjnym języku jak język polski (w którym na dodatek kolejność słów jest dość dowolna), sekwencje form sąsiadujących słów nie wypełniają równomiernie przestrzeni tych sekwencji. Innymi słowy: dużo większa jest szansa że np. po bezokoliczniku nastąpi rzeczownik w bierniku, niż w mianowniku np. we fragmencie zdania: „zrobić błąd" (słowo „błąd" ma zarówno w mianowniku co w bierniku liczby pojedynczej formę „błąd"). Zastosowanie podejścia statystycznego pozwala przypisać rzeczownikowi „błąd" formę „biernik liczby pojedynczej" a odrzucić (jako mniej prawdopodobną) formę „mianownik liczby pojedynczej". Przy czym należy zaznaczyć, że zaimplementowana metoda z natury musi się borykać z następującymi problemami:

1. Konieczne jest zgromadzenie dużych zbiorów tekstów i oznakowanie ich formami gramatycznymi (co jest bardzo pracochłonne)
2. Metoda słabo radzi sobie w przypadku, gdy w wejściowym dokumencie dla danego tokenu nie ma żadnej proponowanej formy
3. Metoda jest zupełnie nieodporna na błędy w dokumencie wejściowym (tj. w przypadku gdy w dokumencie wejściowym dla danego tokenu nie ma właściwej formy, a są inne formy – metoda nie znajdzie właściwej formy)

### Testy

W celu testowania skuteczności usuwania niejednoznaczności w przypisaniu form gramatycznych dla wejściowych plików wykonano następujące przygotowania:

- „Ręcznie" usunięto niejednoznaczności z plików testowych

- Zbudowano 2 modele:
    - mały - na podstawie jednego tomu powieści „Lalka” (model_mini.xml)
    - duży – na podstawie całej powieści (model_maxi.xml)
- Wyniki usuwania niejednoznaczności otrzymane na podstawie zbudowanych modeli porównano z „ręcznie” opracowanymi plikami

Taki sposób testowania, oparty o arbitra ludzkiego, wydaje się być najrozsądniejszy. Nie można jednak zapomnieć o błędzie ludzkim, który mógł się wkraść podczas ręcznego tworzenia pliku na podstawie którego później prowadzono porównania. Testowane zbiory tokenów (pliki z danymi wejściowymi) dobrano w ten sposób, aby możliwe było sprawdzenie jak zachowuje się przyjęty model i algorytm dla danych, które „zna” oraz dla danych zupełnie nowych. Osiągnięto to przez testowanie poprawności dla części tekstu na podstawie, którego tworzony był model (jeden rozdział wyżej wspomnianej powieści – „lalka1.xml”) oraz tekstu pisanego zupełnie innym językiem – tekstu para technicznego – „architekci.xml”.

Wyniki skuteczności usuwania niejednoznaczności w przeprowadzonych testach przedstawia poniższa tabela:

| **liczba tokenów** | **nazwa pliku/rodzaj – określony powyżej** | | | |
|---|---|---|---|---|
| | „*lalka1.xml*” | | „*architekci.xml*” | |
| rodzaj modelu | mały | duży | mały | duży |
| wszystkich | 1725 | | 784 | |
| nierozpoznanych* | 183 | | 105 | |
| niejednoznacznych** | 441 | | 220 | |
| **wariant pracy programu**: | *ignorowanie sekwencji które były budowane w oparciu o nierozpoznane tokeny* | | | |
| dobrze poprawionych | 103 (23.4%) | 120 (27.2%) | 48 (21.8%) | 49 (22.3%) |
| źle poprawionych | 69 (15.6%) | 81 (18.4%) | 13 (5.9%) | 15 (6.8%) |
| niepoprawionych** | 269 (61.0%) | 240 (54.4%) | 159 (72.3%) | 156 (70.9%) |
| **wariant pracy programu** | *uwzględnienie sekwencji które były budowane w oparciu o nierozpoznane tokeny* | | | |
| dobrze poprawionych | 137 (31.1%) | 153 (34.7%) | 75 (34.1%) | 75 (34.1%) |
| źle poprawionych | 95 (21.5%) | 105 (23.8%) | 28 (12.7%) | 30 (13.6%) |
| niepoprawionych** | 209 (47.4%) | 183 (41.5%) | 117 (53.2%) | 115 (52.3%) |

*) nieposiadających opisu atrybutów

**) tokenów o różnych ‘id’ wymagających wykluczenia niejednoznaczności

Otrzymane wyniki nie są zadowalające, co w dużej mierze spowodowane jest ograniczonym zakresem zbioru trenującego. Z drugiej strony w przypadku, gdy budowanie modelu i testy są przeprowadzane na tym samym dokumencie można by oczekiwać wyników znacznie lepszych. Otrzymane wyniki pokazują, że (przynajmniej dla badanych tekstów) zwiększenie ilości tekstu wykorzystanego do konstrukcji modelu poprawiło jakość usuwania niejednoznaczności.

Wyniki pokazują także, że metoda dość dobrze radzi sobie w sytuacjach, gdy w pobliżu tokenu o niejednoznacznie określonej formie gramatycznej znajduje się token o nieprzypisanej żadnej formie gramatycznej (po uwzględnieniu sekwencji z nierozpoznanymi tokenami procent tokenów dobrze rozpoznanych wzrósł).

Ze względu na charakter metody, możliwości rozbudowy i poprawy skuteczności działania programu nie polegają na dopisywaniu reguł czy zmianie konfiguracji (metoda nie bazuje na jawnych regułach i nie wykorzystuje żadnego słownika).

Możliwości konfiguracji i rozbudowy mogą opierać się na:

1. Konstruowaniu modeli na coraz większych zbiorach tekstów (bez zasadniczej zmiany działania samego modelu) – ten sposób jest najbardziej naturalny dla metody. Intuicja (potwierdzona wynikami testów) podpowiada, że gromadzenie coraz większych, bardziej przekrojowych tekstów, i konstruowanie w oparciu o nie modeli powinno podnosić jakość zwracanych wyników (kwestią otwartą jest jak dobre rezultaty można osiągnąć, stosując jedynie takie rozwijanie programu)
2. Modyfikacja w sposobie obsługi sytuacji, gdy w modelu nie znalazła się żadna pasująca forma gramatyczna – tzn. kiedy model nie potrafi zaproponować żadnej formy gramatycznej, która znajdowałaby się pośród proponowanych w danych wejściowych. Modyfikacja taka mogłaby korzystać z modelu:
    a. wybierając np. najwyżej punktowaną możliwość z modelu, pomimo tego, że wybrana w ten sposób forma nie występuje wśród form proponowanych w danych wejściowych (takie rozwiązanie uniezależniłoby częściowo od błędów w danych wejściowych – w postaci braku właściwej formy wśród proponowanych)

b. wybierając taką formę spośród proponowanych przez dane wejściowe, która byłaby bliska którejś z proponowanych przez model. Aby to zrealizować konieczne byłoby zdefiniowanie miary odległości między różnymi formami gramatycznymi (tak, żeby np. formy czasownika różniące się tylko rodzajem były bliżej niż czasownik w jednej z tych form od przysłówka lub rzeczownika). Takie rozwiązanie mogłoby uniezależnić algorytm od „braków" w modelu.

3. Wykorzystanie dodatkowo reguł eksperckich. Reguły takie mogłyby być przygotowywane wcześniej lub tworzone poprzez interakcję z operatorem w trakcie przetwarzania. W sytuacjach, gdy dotychczasowe reguły są niewystarczające, operator podejmowałby decyzję, a program uzupełniałby model.

## 3.5. Rozpoznawanie nazw własnych

**Rozpoznawanie nazw własnych** (ang. ***Named Entity Recognition***, ***NER***) polega na stwierdzeniu, że dane słowo lub ciąg słów (fraza) jest nazwą własną i jednocześnie przyporządkowanie jej odpowiedniej kategorii i ewentualnie podkategorii, np.:

- organizacja
- osoba
- lokalizacja
- czas (data, godzina)
- ilość (liczba, kwota pieniężna, procent)
- itp.

Nazwy własne bardzo często występują w dokumentach tekstowych. Przykładowo według [Coates-Stephens, 1992] notatki prasowe mogą zawierać do 10% nazw własnych. Poprawne rozpoznanie nazw własnych może mieć istotny wpływ na wynik dalszego przetwarzania w wielu zagadnieniach IE, IR oraz TDM, np. kategoryzacji dokumentów.

Zadanie rozpoznawania nazw własnych zostało zdefiniowane jako jedno z zagadnień IE na konferencjach MUC (Patrz rozdział 1.3.1). W ogólnym przypadku zadanie to może być bardzo złożone. Rozważmy następujące („fikcyjne") zdanie:

*W miniony wtorek na zgromadzeniu wspólników Gores S.A. została przedstawiona przez prezesa Juliana Kosmulaka propozycja podjęcia współpracy z Roman Sp. z o.o.*

Przy pełnej realizacji zadania w powyższym zdaniu powinny być rozpoznane następujące frazy:

- Czas: *W miniony wtorek*
- Lokalizacja: *zgromadzenie wspólników*
- Nazwa (organizacja): *Gores S.A.*
- Nazwa (osoba): *Julian Kosmulak*
- Nazwa (organizacja): *Roman Sp. z o.o.*

Określenie czy dana fraza jest nazwą własną i do jakiej kategorii należy może zależeć zarówno od jej treści i struktury, jak również od kontekstu. Przykładowo skrót *S.A.* lub *Sp. z o.o.* wskazują, że wcześniej pojawia się nazwa firmy. Jednak słowo *Roman* jest jednocześnie imieniem. W tym przypadku usunięcie niejednoznaczności jest proste. Jednak w przypadku ogólnym może to być złożony problem i może wymagać wykorzystania szerokiej wiedzy z dziedziny, z której pochodzi analizowany tekst.

Najprostszą strategią rozpoznawania nazw własnych jest zastosowanie odpowiednich słowników. Jednak pewne kategorie nazw własnych, np. nazwy firm, są zbyt liczne, aby umieścić je w słowniku. Dodatkowo mogą się pojawiać nowe tego typu nazwy. Oprócz tego mogą się one pojawiać w wielu różnych postaciach, np. w formie skróconej. Z tego powodu bezpośrednie stosowanie słownika może okazać się w wielu wypadkach niewystarczające.

Warto również zauważyć, że w przypadku języków fleksyjnych, jakim jest język polski, większość nazw własnych podlega odmianie, w tym np. imiona i nazwiska. Co prawda w większości wypadków nazwa własna odmienia się tak jak rzeczownik o identycznej końcówce. Jednak dodatkowo utrudnia to prawidłowe rozpoznawanie i jest źródłem potencjalnych błędów przy znajdywaniu formy podstawowej.

Większość systemów IE biorących udział w rozpoznawaniu nazw własnych podczas konferencji MUC-7 bazowało na zestawach reguł kontekstowych napisanych przez ekspertów [Wakao, 1996]. W [Gallippi, 1996] zaproponowano strategię używającą przygotowane przez projektanta wzorce do rozpoznawania fraz oraz drzewa decyzyjne do klasyfikowania nazw własnych. W ostatnim czasie zostało zaproponowanych wiele metod z zakresu uczenia się maszyn w celu rozpoznawania nazw własnych. [Bikel, 1997] zaprezentował statystyczne podejście adaptacyjne korzystające z wersji standardowego ukrytego modelu Markova. Metody oparte o maksymalną entropię zostały zaprezentowane w [Borthwick, 1999].

### Implementacja

W [Proj7] zaproponowano i zaimplementowano algorytm rozpoznawania nazw własnych. Program pobiera dane wejściowe w formacie XML zgodnym z opisem w rozdziale 6.1. „Format zapisu tokenów” (str. 134). Program realizuje przetwarzanie tekstu wejściowego w oparciu o zdefiniowany zestaw informacji dodatkowych. Takie informacje to przede wszystkim:

- reguły określające sposób rozpoznawania i klasyfikowania nazw własnych,
- obszerne słowniki nazw własnych pogrupowanych wedle typu nazw.

Przetwarzanie odbywa się zdaniami. Dla każdego kolejnego słowa w zdaniu sprawdzane jest dopasowanie do reguł i słowników. Na tej podstawie określany jest typ i podtyp nazwy własnej a także ustawiane jest id nazwy własnej, które w kolejnej iteracji jest wykorzystywane do rozpoznawania nazw wielowyrazowych.

Program wykorzystuje szereg plików pomocniczych zawierających reguły i słowniki:

- dict.xml – plik zawierający reguły rozpoznawania nazw własnych. Zawiera elementy odpowiadające poszczególnym typom nazw własnych, które z kolei zawierają podelementy odpowiadające ścieżkom do plików słowników dla tych typów nazw, bądź też oparte są na wyrażeniach regularnych reguły rozpoznawania danego typu nazwy własnej.
- pliki słownika – pliki tekstowe. Każda nazwa własna zapisana jest w osobnej linii.

Wynikowy plik XML zawiera następujące atrybuty w tokenach:

- nazwatyp – zostanie ustawiony na typ nazwy własnej w przypadku, gdy token zostanie rozpoznany jako nazwa własna, bądź jej część. Wartości, jakie może przyjąć atrybut (zgodnie z zaproponowanym formatem XML wartości nie zawierają polskich liter):
    - osoba
    - miejsce
    - czas
    - waluta
    - liczba
    - tytul
    - samochod
    - swieto
    - instytucja
    - nieokreslony
- nazwapodtyp – zostanie ustawiony na podtyp danego typu nazwy własnej w przypadku, gdy token zostanie rozpoznany jako nazwa własna, bądź też część nazwy własnej, oraz gdy rozpoznany typ nazwy własnej posiada podtypy. Wartości, jakie może przyjąć atrybut nazwapodtyp, to, w zależności od wartości atrybutu nazwatyp odpowiednio (zgodnie z zaproponowanym formatem XML wartości nie zawierają polskich liter):
    - osoba
        - imie
        - nazwisko
        - tytul
        - narodowosc
        - zawod
    - miejsce
        - kraj
        - miejscowosc
    - czas
        - data
        - godzina

        - dzien
        - miesiac
    - waluta
        - kod
        - waluta
    - liczba
        - rzeczywista
        - naturalna
        - procent
        - symbol
    - samochod
        - marka
        - model
    - instytucja
        - szkola
        - firma
        - inna
    - nieokreslony
        - slowoDuze
        - slowoMieszane
        - slowoPierwszaDuza

### Testy

Implementację przetestowano na wybranych plikach stanowiących wynik działania programów realizujących wcześniejsze etapy przetwarzania (wyniki z [Proj6]). Pliki te nie poddawane były żadnym modyfikacjom przed podaniem na wejście programu. Liczbę występujących w tych plikach nazw własnych oraz skuteczność rozpoznawania nazw własnych określono manualnie. Skuteczność jest określana jako stosunek prawidłowo rozpoznanych nazw własnych do liczby wszystkich nazw własnych występujących w dokumencie. Wyniki przedstawia poniższa tabela.

| Plik | liczba nazw własnych | liczba nazw własnych rozpoznanych prawidłowo | skuteczność |
|---|---|---|---|
| Architekci4.xml | 88 | 45 | 51% |
| Bankowa superfuzja - Unicredito przejmuje HVB.xml | 81 | 40 | 49% |
| Dokąd zmierza chińska nauka.xml | 15 | 7 | 47% |

Jak widać w powyższej tabeli skuteczność rozpoznawania nazw własnych waha się w granicach 47%-51%. Należy podkreślić, że w niektórych przypadkach niepoprawne rozpoznanie nazwy wynikało z błędnych danych wejściowych, a więc niepoprawnym podziale na zdania lub błędnie rozpoznanej części mowy.

Podsumowując, rozpoznawanie nazw własnych w języku polskim jest w ujęciu ogólnym zadaniem złożonym. Znaczenie słów jest uzależnione od kontekstu zdania. Dodatkowym utrudnieniem jest fleksja obejmująca również nazwy własne. Poprawiając wyniki wcześniejszych etapów analizy oraz rozszerzając zakres słowników i reguł wykrywania na pewno można poprawić skuteczność rozpoznawania nazw własnych. Skuteczność rozpoznawania można znacząco zwiększyć dostosowując słowniki – uzupełniając je o słowa występujące w przetwarzanym tekście. Odpowiednio wielkie słowniki pozwoliłyby na osiągnięcie znacznie większej skuteczności średniej – liczonej z większej ilości tekstów. Jednak ze względu na dużą swobodę tworzenia nazw własnych w języku (np. nazw firm) trudno spodziewać się uzyskania 100% skuteczności. Wyniki na poziomie 70%-80% należałoby uznać za satysfakcjonujące.

## 3.6. Zastępowanie zaimków

W języku naturalnym bardzo często używane są zaimki. Pozwalają one na szybsze wyrażanie myśli i upraszczanie wypowiedzi. Powodują także nawiązywanie do kontekstu wypowiedzi, w szczególności do zdań wcześniejszych. W przypadku analizy całości tekstu jest on zrozumiały. W przypadku analizy STP bardzo przydatne może okazać się jednak zastępowanie zaimków. Polega to na zastępowaniu zaimka słowem, do którego on się odnosi. Dzięki temu zdania w dużym stopniu zachowują swoje znaczenie, także po ich wyjęciu z kontekstu, co wyraźnie upraszcza dalszą analizę.

Najtrudniejszym elementem tego zadania jest poprawne ustalenie słowa lub wyrażenia, do którego odnosi się dany zaimek. Niestety czasami zaimek wskazuje na obiekt, który jest przedstawiony w tekście bardzo opisowo. Np.:

*Spotkał tego, który nie wie co to strach. Nie znał go wcześniej.*

W powyższym przykładzie zaimek *go* odnosi się do wcześniejszego zaimka *tego*, który z kolei opisany jest całym zdaniem podrzędnym. W pewnych sytuacjach określenie do czego odnosi się zaimek może być nawet niemożliwe, gdyż obiekt wskazywany może w ogóle nie wystąpić w tekście, lecz może wynikać z danej dziedziny lub np. środowiska kulturowego. Zadania zastępowania zaimków jest jednak w większości wypadków realizowalne z zachowaniem wymienionych ograniczeń.

Zagadnienie zastępowania zaimków wiąże się ściśle z rozpoznawaniem nazw własnych omówionym w rozdziale. 3.5. Rozpoznawanie nazw własnych (str. 79).

Problem ten występuje także w przypadku zagadnienia omówionego w rozdziale 3.7 Rozkład zdań złożonych na zdania proste (str. 89).

Pod względem zagadnienia zastępowania zaimków można przyjąć następujące sposoby klasyfikacji zaimków.

Ze względu na zastępowaną część mowy:

- rzeczowne (*ty, się, kto, nic*)
- przymiotne (*mój, ten, który, jakiś*)
- liczebne (*tyle, ile, ileś*)
- przysłowne (*tam, kiedy, gdzieś, nigdy*)

Ze względu na znaczenie:

- osobowe (*Ja czekam; Oni przyszli*)
- zwrotne (*Kasia się myje; Uświadomiłem to sobie*)
- dzierżawcze (*Daj mi mój ołówek; Daj mi swój ołówek*)
- wskazujące (*Daj mi ten ołówek; Połóż go tutaj*)

- pytające (*Kto to zrobił? Kiedy to się stało?*)
- względne (*Kto zawinił, zostanie ukarany; Czego nie wiesz, tego sobie nie przypomnisz*)
- nieokreślone (*Byłem tu kiedyś; Coś mi się przypomina*)
- przeczące (*Nikt mi o tym nie mówił; Nigdy tam nie byłem*)
- upowszechniające (*Wszyscy dziś przyszli; Zawsze o tobie pamiętam; Wszędzie tego szukałem*)

Dopasowanie zaimków w języku polskim można sprowadzić do następujących zasad:

1. Jeżeli zaimek odmienia się przez rodzaje, to pasujący wyraz również musi odmieniać się przez rodzaje i być tego samego rodzaju.
2. Jeżeli zaimek ma liczbę (pojedynczą, mnogą), to pasujący wyraz musi mieć taką samą liczbę.
3. Zachowana musi być zgodność kategorii znaczeniowej:
    - zaimek osobowy (np. „*on*”) i dzierżawczy (np. „*jego*”) pasuje do wyrazu określającego osobę,
    - zaimek przysłowny określający lokalizację (np. „*tam*”) pasuje do nazwy lokalizacji,
    - zaimek przysłowny określający czas (np. „*wtedy*”) pasuje do określenia czasu,
    - zaimek liczebny pasuje do liczby, procentu lub kwoty pieniężnej.

#### Implementacja

W [Proj7] zaimplementowano algorytm dopasowywania zaimków. Program pobiera dane wejściowe w formacie XML zgodnym z opisem w rozdziale 6.1. Format zapisu tokenów (str. 134). Program poszukuje tokeny (rzeczowniki, bądź nazwy własne) w oparciu o zestaw dostępnych reguł przechowywanych w słowniku (plik

pronounRules.xml). Każdy element słownika reguł zawiera słowo – zaimek, jego przypadek, liczbę i rodzaj, kierunek przeszukiwania słów dopasowanych i maksymalną ilość zdań, w jakiej należy poszukiwać dopasowanego tokenu.

Zasada przetwarzania jest następująca:
Program przegląda tekst wejściowy w poszukiwaniu tokenów sklasyfikowanych jako zaimki. W momencie odnalezienia zaimka przeglądane są reguły w poszukiwaniu takich, które pasują do danego zaimka. Dla każdej reguły, która odpowiada słowu – zaimkowi, liczona jest miara dopasowania, która mierzy jak dobrze dana reguła pasuje do rozpatrywanego zaimka. Miara uwzględnia:

- rodzaj zaimka
- liczbę zaimka
- przypadek zaimka

Jeżeli reguła posiada któryś z uwzględnianych elementów sprzeczny z zaimkiem, wtedy natychmiast jest odrzucana. W przeciwnym razie miarą dopasowania jest liczba uwzględnionych atrybutów zgodnych z rozpatrywanym zaimkiem (zakres [0-3]).

Gdy już znane są miary dopasowania reguł, do dalszej analizy wybierane są te, które mają największą wartość miary dopasowania. Następnie dla każdej reguły, w zależności od kierunku poszukiwania dla niej zdefiniowanego, rozpoczyna się przegląd tekstu wejściowego od rozpatrywanego zaimka w lewo lub w prawo. Przeglądanie tekstu odbywa się w zakresie zdań odległych maksymalnie o liczbę określoną w regule od zdania zawierającego rozpatrywany zaimek. Każdy napotkany rzeczownik badany jest pod kątem zgodności z rozpatrywaną regułą. W zależności od stopnia zgodności, pod względem rodzaju i liczby. Dodatkowo miara zgodności uwzględnia odległość rzeczownika od zaimka (stosowany jest tu ustalony eksperymentalnie modyfikator). Oprócz rzeczowników dopasowywane są też nazwy własne, które otrzymują ustalony eksperymentalnie stały współczynnik dopasowania. Współczynnik ten również podlega modyfikacji uwzględniającej odległość od zaimka.

Spośród wszystkich dopasowanych tokenów wybierany jest najbliższy zaimkowi spośród tych o maksymalnej mierze dopasowania.

### Testy

Implementację przetestowano na wybranych plikach stanowiących wynik działania programów realizujących wcześniejsze etapy przetwarzania (wyniki z [Proj6]). Pliki te nie poddawane były żadnym modyfikacjom przed podaniem na wejście programu. Skuteczność określono poprzez manualne sprawdzenie wyników. Podczas liczenia skuteczności dopasowania zaimków uwzględniane były tylko tokeny, które rzeczywiście były zaimkami. Przykładowo w danych wejściowych występował błąd polegający na opisaniu jako zaimek czasownika *mieć* w 3 osobie liczby pojedynczej (*ma*). Takie przypadki nie były liczone. Jako poprawne uwzględniane były również sytuacje braku dopasowania zaimka, jeśli takie dopasowanie nie było możliwe z powodu braku wystąpienia w tekście zastępowanego słowa. Przykładem może być zaimek *ja* w dialogach, czy też np. zaimek *wszyscy*. Wyniki testów przedstawia poniższa tabela.

| **Plik** | **liczba zaimków** | **liczba zaimków dopasowanych prawidłowo** | **skuteczność** |
|---|---|---|---|
| Architekci4.xml | 13 | 10 | 77% |
| Bankowa superfuzja - Unicredito przejmuje HVB.xml | 17 | 8 | 47% |
| lalka1.xml | 92 | 38 | 41% |
| MAGNETYZER.xml | 61 | 29 | 47% |
| Mechanizm Roznicowy.xml | 20 | 8 | 40% |
| Naglowki.xml | 93 | 34 | 36% |
| Opole 2005 - Tylem do sceny.xml | 46 | 16 | 35% |
| Pierwsze dziecko z przeszczepu jajnika innej kobiety.xml | 24 | 12 | 50% |
| seleukos.xml | 72 | 31 | 43% |
| we_test.xml | 6 | 0 | 0% |

Jak widać w powyższej tabeli skuteczność dopasowywania zaimków wykazuje duże wahania w zależności od konkretnego dokumentu, jednak generalnie utrzymuje się na poziomie nieco ponad 40%. Istotnym elementem ograniczającym skuteczność dopasowania zaimków są nieprawidłowości występujące na wcześniejszych etapach analizy. Poprawienie jakości danych wejściowych powinno zwiększyć skuteczność dopasowania zaimków.

Podsumowując, dopasowanie i zastępowanie zaimków w języku polskim jest zagadnieniem skomplikowanym z powodu swobodnego szyku zdań. Z drugiej strony fleksja występująca w języku polskim pozwala na uwzględnianie wielu atrybutów w procesie dopasowania, co daje dosyć duże możliwości odnalezienia wskazywanego słowa. Niezwykle ważna jest również poprawność danych tekstowych. Prawidłowe rozpoznanie części mowy i opatrzenie ich prawidłowymi atrybutami znacznie poprawiłoby osiąganą skuteczność.

## 3.7. Rozkład zdań złożonych na zdania proste

Jednym z ciekawych zadań analizy tekstu jest rozkład zdań złożonych i wielokrotnie złożonych na zdania proste. Z pewnym przybliżeniem można przyjąć, że każde zdanie proste zawiera konkretną informację (fakt). Tak więc zadanie to jest pewną aproksymacją problemu ekstrakcji faktów z treści dokumentu tekstowego.

Uzyskanie wyłącznie zdań prostych pozwala na uproszczenia w dalszych fazach analizy. W przypadku IE liczba reguł i poziom ich skomplikowania mogą być niższe przy założeniu występowania wyłącznie zdań prostych. W wielu rozwiązaniach problemu sumaryzacji występuje wybieranie zdań, które będą umieszczone w streszczeniu. Im te zdania są krótsze, tym krótsze i bardziej zwięzłe może być streszczenie.

Z problemem tym powiązane jest również zagadnienie zastępowania zaimków. Omówiono je w rozdziale 3.6 (str. 84).

Aby przyjrzeć się bliżej problemowi rozkładu zdań złożonych na proste należy przeanalizować występujące w języku polskim rodzaje zdań złożonych.

**1) zdania złożone współrzędnie**

a) **łączne** – Zdania te są połączone przy pomocy spójników: *i, a , oraz, tudzież, to* lub połączenie jest bezspójnikowe (wówczas występuje przecinek).

Są to zdania typu:

*Ala ma kota i ma psa. → Ala ma kota. Ala ma psa.*

*Ala ma kota i ona ma psa. → Ala ma kota. Ala ma psa.*

*Ala ma kota i Ala ma psa. → Ala ma kota. Ala ma psa.*

W zdaniach tego typu w drugim zdaniu może wystąpić zaimek osobowy, który należy zamienić odpowiednim rzeczownikiem. Nie ma wtedy potrzeby wstawiania podmiotu – jest on już wstawiony przez zamianę zaimka osobowego.

b) **rozłączne** (**alternatywne**) – Rozkład tego typu zdań powoduje utratę znaczenia. Przykładowe zdanie poniżej po podziale utraciło sens.

*Albo ludzie przestaną prowadzić wojny, albo znikną z powierzchni Ziemi*.

*Ludzie przestaną prowadzić wojny. Ludzie znikną z powierzchni Ziemi.*

c) **wyłączające** - treści obu zdań wykluczają się nawzajem (żadna nie może dojść do skutku). Sytuacja jest podobna jak w poprzednim podpunkcie.

d) **przeciwstawne** – połączone za pomocą spójników przeciwstawnych – *a, ale, jednak, zaś, wszakże, jednak, jednakże, tylko, przeciwnie, natomiast, tymczasem* lub bezspójnikowo. Np.: *Powiedział jej o tym, lecz mu nie uwierzyła*. W tym przypadku rozkład zdania jest prosty. Podział następuje w miejscu spójnika (lub przecinka).

e) **wynikowe** – połączone za pomocą spójników: *więc, zatem, toteż, przeto, dlatego, tedy, wobec tego.* Sytuacja analogiczna do pkt. a). Z tym jednak, że następuje utrata informacji o implikacji faktów. Jednak w pewnych sytuacjach jest to akceptowalne, np.:

*Ala ma kota, więc ma dużo sprzątania → Ala ma kota. Ala ma dużo sprzątania.*

f) **synonimiczne** (**włączne**) – Drugie zdanie wyraża tę samą treść co zdanie pierwsze, lecz inaczej ujętą. Zdania są połączone za pomocą wyrazów: *czyli, mianowicie, to jest (tj.), to znaczy (tzn.), inaczej, innymi słowy, słowem* lub bezspójnikowo. Sytuacja analogiczna do pkt. e). Np.:

*Ala ma kota, czyli ma czworonoga → Ala ma kota. Ala ma czworonoga.*

**2) zdania złożone podrzędnie**

a) **zdania podmiotowe** – Rozkład zdań na proste jest w tym przypadku problematyczny.

*Wszystkim zdawało się, że Wojski wciąż gra jeszcze.*

b) **zdania orzecznikowe** – Rozkład zdań na proste jest w tym przypadku problematyczny. *Jadzia jest tym dla nas, czym matka jest dla dzieci.*

*Był taki, jak jest zawsze*.

c) **zdania dopełnieniowe** – zastępuje dopełnienie zadania nadrzędnego. Odpowiada na pytania przypadków zależnych: *Kogo? Czego? Komu? Czemu? Kogo? Co? Kim? Czym? (o) kim? (o) czym?*. Rozpoczyna się od zaimków względnych: *co, jak, gdzie, kiedy dlaczego,* spójników: *że, iż, żeby, ażeby, aby, by, jakby*, oraz partykuły *czy*. Rozkład jest problematyczny.

*Bał się, by nie zbłądzić.*

d) **przydawkowe** – zastępuje przydawkę zdania nadrzędnego. Odpowiada na pytania przydawki: *Jaki? Który? Czyj? Ile?* Rozpoczyna się od zaimków względnych: *co, jaki, który, kiedy*, spójników: *że, iż, ażeby, aby, by jakoby.* Te zdania są zamieniane, gdyż zdanie podrzędne określa przydawkę nadrzędnego, która staje się podmiotem zdania podrzędnego. Przykładami mogą być:

*Piotr chodzi do szkoły, która jest ładna → Piotr chodzi do szkoły. Szkoła jest ładna.*

Tutaj zdanie podrzędne opisywało „szkołę" – wiadomo na co wskazywał czasownik – „*ładna*", więc podmiotem drugiego zdania musi być „*szkoła*".

*Piotr chodzi do szkoły, której on nie lubi*. → *Piotr chodzi do szkoły. Szkoły Piotr nie lubi.*

Tutaj czasownik „*lubi*" odnosił się do Piotra, więc podmiotem musi być „*Piotr*".

e) **okolicznikowe** – zastępują okoliczniki zdania nadrzędnego, odpowiadają na te same pytania co okoliczniki. W szczególnych przypadkach można dokonać rozkładu. Wiąże się to jednak często z utratą początkowego sensu zdania.

*Tam gdzie drwa rąbią, tam wióry lecą.* → *Drwa rąbią. Wióry lecą.*

f) **rozwijające** – nie zastępuje części zdania, rozwija treść zdania. Zdania są połączone zaimkiem względnym: *co, czego, który, gdzie, kiedy*. Rozkład jest możliwy, jednak pełne przekazanie informacji może okazać się skomplikowane.

*Nie jesteś uczciwy, co nas bardzo smuci.* → *Nie jesteś uczciwy. To bardzo nas smuci.*

*Wydajność systemu spadła, co zwiększyło czas oczekiwania.* → *Wydajność systemu spadła. To zwiększyło czas oczekiwania.*

Taki rozkład nie powoduje, że drugie zdanie nie może wystąpić samodzielnie. Stworzenie niezależnych zdań jest procesem skomplikowanym i wymaga stosowania złożonych przekształceń lingwistycznych, jak np. tworzenie rzeczowników od innych części mowy.

*Nie jesteś uczciwy, co nas bardzo smuci.* → *Nie jesteś uczciwy. Bardzo nas smuci twoja nieuczciwość.*

*Wydajność systemu spadła, co zwiększyło czas oczekiwania.* → *Wydajność systemu spadła. Spadek wydajności systemu zwiększył czas oczekiwania.*

Zdanie rozwijające przekazuje zwykle mniej istotną treść od treści zawartej w zdaniu rozwijanym. Stąd w zadaniu sumaryzacji możliwe jest zastosowanie podejścia heurystycznego polegającego na pomijaniu zdań rozwijających.

W ramach prac badawczych opracowano algorytmy rozkładu zdań złożonych na proste oparte o zestawy reguł i zrealizowano implementacje na nich oparte. Rozwiązania te bazują na wykrywaniu przecinków i łączników zdań oraz stosowaniu odpowiedniego zbioru reguł przetwarzania. Eksperymenty przeprowadzone w [Proj9] potwierdzają możliwość praktycznego wykorzystania tych rozwiązań. Implementacja ta pobiera dane wejściowe w postaci tokenów zawierających informacje morfologiczne i syntaktyczne. Jak widać program radzi sobie nawet ze zdaniami wielokrotnie złożonymi.

| Zdanie wejściowe | Zdania wyjściowe |
|---|---|
| *Ala ma kota i cieszy się, lecz ma psa.* | *Ala ma kota. Ala cieszy się. Ala ma psa.* |
| *Ala ma kota i ona cieszy się.* | *Ala ma kota. Ala cieszy się.* |
| *Piotr jest uczniem i chodzi do szkoły* | *Piotr jest uczniem. Piotr chodzi do szkoły.* |
| *On lubi się uczyć.* | *Piotr lubi się uczyć.* |
| *Piotr jest uczniem i on chodzi do szkoły.* | *Piotr jest uczniem. Piotr chodzi do szkoły.* |
| *Piotr chodzi do szkoły, która jest ładna.* | *Piotr chodzi do szkoły. Szkoła jest ładna.* |
| *Piotr chodzi do szkoły, której nie lubi.* | *Piotr chodzi do szkoły. Szkoły Piotr nie lubi.* |
| *Gdzie drwa rąbią, tam wióry lecą.* | *Drwa rąbią. Wióry lecą.* |
| *Tam gdzie drwa rąbią, wióry lecą.* | *Drwa rąbią. Wióry lecą.* |

Uzyskane wyniki były zachęcające. W [Proj10] stworzono implementację, która opiera się o wykorzystanie wyrażeń regularnych, co ma na celu łatwiejsze rozbudowywanie zbioru reguł. Zaimplementowano również operację odwrotną, czyli łączenie zdań prostych w złożone. Przeprowadzone też pełniejsze testy na danych rzeczywistych, będących efektem wcześniejszych etapów analizy.

Koncepcja zastosowanego podejścia wygląda następująco. Każde zdanie wczytane z wejścia, zapisane w odpowiednio zdefiniowanym formacie XML, jest dopasowywane do kolejnych wzorców zdań złożonych, zapisanych w postaci wyrażeń regularnych. W przypadku dopasowania, wykonywana jest operacja zastępowania wyrażenia regularnego innym odpowiednim wzorcem. Wyrażenia regularne są tak skonstruowane, że w rezultacie otrzymujemy dwa zdania, z których pierwsze jest zdaniem prostym. Na drugim z nich rekurencyjnie wykonywana jest ta sama procedura.

Dla przykładu, wyrażenie rozpoznające zdania złożone współrzędne przeciwstawne z 1 podmiotem wygląda następująco:

```
L"\\A"
L"(?<p0>(^\\s*<token[^/]*czescMowy=\"(?!czasownik)[^\"]+\"[^/]*/>\\s*$.)*)"
L"(?<rzecz>^\\s*<token[^/]*czescMowy=\"((rzeczownik)|(przymiotnik))\"[^/]*/>\\s*$.)"
L"(?<p1>(^\\s*<token[^/]*czescMowy=\"(?!czasownik)[^\"]+\"[^/]*/>\\s*$.)*)"
L"(?<czas1>^\\s*<token[^/]*czescMowy=\"czasownik\"[^/]*/>\\s*$.)"
L"(?<p2>(^\\s*<token[^/]*czescMowy=\"(?!czasownik)[^\"]+\"[^/]*/>\\s*$.)*)"
L"(?<przec>^\\s*<token[^/]*slowo=\",\"[^/]*/>\\s*$.)"
L"(?<spojnik>^\\s*<token[^/]*slowo=\"((ale)|(lecz))\"[^/]*/>\\s*$.)"
L"(?<p3>(^\\s*<token[^/]*czescMowy=\"(?!czasownik)[^\"]+\"[^/]*/>\\s*$.)*)"
L"(?<czas2>^\\s*<token[^/]*czescMowy=\"czasownik\"[^/]*/>\\s*$.)"
```

*\\A* oznacza, że szukamy od początku zdania. Zdanie zostaje podzielone na fragmenty. <*p0*> oznacza grupę słów nie będących czasownikiem. Potem występuje czasownik <*czas1*>, potem grupa słów nie będących czasownikiem <*p1*>. Następnie występuje przecinek <*przec*>, i spójnik. Następnie występuje grupa słów niebędących czasownikiem <*p2*> i czasownik <*czas2*>.

Wykorzystanie wyrażenia oznaczającego grupę słów niebędących czasownikiem zapobiega błędnemu dopasowaniu wzorca w przypadku zdań wielokrotnie złożonych. Użycie *\\A* znacznie przyspiesza poszukiwania, a także gwarantuje, że pierwsze zdanie otrzymane po podziale będzie zdaniem prostym.

Wzorce „kompilowane" są na początku wykonania programu, a następnie wykorzystywane są już wersje skompilowane jako obiekt Regex ze standardowej biblioteki .NET.

W przypadku dopasowania powyższego wzorca, dalsze poszukiwanie wzorców jest przerywane i znalezione wyrażenie jest przetwarzane na następujące:

```
L"${p0}${rzecz}${p1}${czas1}${p2}"
L"    <token id=\"X\" slowo=\".\" rodzajTokena=\"znak\"/>\r\n"
L"<sentence_separator/>"
L"${rzecz}${p3}${czas2}"
```

Pierwsza część zdania złożonego bez zmian. Następnie pomijany jest przecinek i spójnik i wstawiana jest kropka. Atrybut ID tokenu z kropką jest ustawiany jako X (nieznany). Następnie wstawiany jest tymczasowy znacznik <*sentence_separator/*>.

Za nim znajduje się druga część zdania złożonego, która staje się nowym zdaniem prostym. Następnie na drugim otrzymanym zdaniu rekurencyjnie wykonywana jest ta sama procedura i <*sentence_separator/*> jest zastępowany znacznikami początku i końca zdania.

Rozkładane są następujące typy zdań:

1. Proste bez orzeczenia – wyjątek, tj. nie jest rozkładane
2. Proste – wyjątek
3. Podrzędne przydawkowe.
4. Podrzędne okolicznikowe czasu lub miejsca
5. Podrzędne okolicznikowe czasu lub miejsca, odwrócone
6. Podrzędne orzecznikowe – wyjątek
7. Podrzędne dopełnieniowe – wyjątek
8. Współrzędne przeciwstawne i wynikowe lub podrzędne okolicznikowe przyczyny i celu z 2 podmiotami
9. Współrzędne przeciwstawne i wynikowe lub podrzędne okolicznikowe przyczyny i celu z 1 podmiotem
10. Współrzędne przeciwstawne i wynikowe lub podrzędne okolicznikowe przyczyny i celu bez podmiotu
11. Współrzędne rozłączne z 2 podmiotami – wyjątek
12. Współrzędne rozłączne z 1 podmiotem – wyjątek
13. Współrzędne rozłączne bez podmiotu – wyjątek
14. Współrzędne łączne i pozostałe złożone z 2 podmiotami
15. Współrzędne łączne i pozostałe złożone z 1 podmiotem
16. Współrzędne łączne i pozostałe złożone bez podmiotu

W przetwarzaniu odwrotnym wczytywane jest kolejne zdanie i wraz z poprzednim wczytanym zdaniem dopasowywane jest do kolejnych wzorców „dwóch zdań, które można połączyć". W przypadku dopasowania, wykonywana jest operacja zastępowania wyrażenia regularnego innym odpowiednim wzorcem. Otrzymane zdanie jest wykorzystywane w kolejnym kroku iteracji wraz ze zdaniem kolejno wczytanym.

Dwa kolejne zdania są łączone łącznikiem <sentence_separator/> i tak utworzony ciąg jest porównywany do wzorca. Na przykład zdania, które można połączyć do zdania złożonego współrzędnego łącznego z jednym podmiotem, są opisane następującym wzorcem:

```
L"\\A"
L"(?<p0>(^\\s*<token[^/]*czescMowy=\"(?!czasownik)[^\"]+\"[^/]*/>\\s*$.)*)"
L"(?<rzecz1>^\\s*<token[^/]*slowo=\"(?<slowo>[^\"]*)\"[^/]*czescMowy=\"((rzecz
ownik)|(przymiotnik))\"[^/]*/>\\s*$.)"
L"(?<p1>(^\\s*<token[^/]*czescMowy=\"(?!czasownik)[^\"]+\"[^/]*/>\\s*$.)*)"
L"(?<czas1>^\\s*<token[^/]*czescMowy=\"czasownik\"[^/]*/>\\s*$.)"
L"(?<p2>(^\\s*<token[^/]*czescMowy=\"(?!rzeczownik)(?!przymiotnik)[^\"]+\"[^/]*/>\\s*$.)
*)"
L"(?<kropka>^\\s*<token[^/]*slowo=\".\"[^/]*/>\\s*$.)"
L"(?<sep>^\\s*<sentence_separator/>\\s*$.)"
L"(?<rzecz2>^\\s*<token[^/]*slowo=\"\\k<slowo>\"[^/]*czescMowy=\"((rzeczownik)|(przymiot
nik))\"[^/]*/>\\s*$.)"
L"(?<p3>(^\\s*<token[^/]*czescMowy=\"(?!czasownik)[^\"]+\"[^/]*/>\\s*$.)*)"
L"(?<czas2>^\\s*<token[^/]*czescMowy=\"czasownik\"[^/]*/>\\s*$.)"
```

W powyższym wzorcu, w grupie *<rzecz2>* następuje odwołanie do grupy o nazwie *<slowo>* znalezionej w grupie *<rzecz1>*. W ten sposób poszukujemy zdań o takim samym podmiocie. Grupa *<p2>* zawiera słowa niebędące rzeczownikami ani przymiotnikami. Dzięki tej heurystyce możliwe jest składanie zdań wielokrotnie złożonych: *<rzecz1>* będzie podmiotem, *<czas1>* pierwszym orzeczeniem, a *<p2>* będzie zawierał ewentualne kolejne orzeczenia, nie będzie mógł jednak zawierać żadnego podmiotu, zatem można wnioskować, że *<rzecz1>* jest ich podmiotem, a zatem jeśli kolejne zdanie również zawiera to samo słowo jako podmiot, to można te dwa zdania połączyć w zdanie współrzędne łączne. Zdania pasujące do tego wzorca są zastępowane następującym zdaniem:

```
L"${p0}${rzecz1}${p1}${czas1}${p2}"
L"\t\t<token id=\"X\" slowo=\"i\" rodzajTokena=\"slowo\" lemat=\"i\"
czescMowy=\"spojnik\"/>\r\n"
L"${p3}${czas2}"
```

Łączenie zdań obejmuje następujące przypadki:

- dwa zdania proste o tym samym podmiocie -> współrzędne łączne z jednym podmiotem,
- dwa zdania proste, z których przydawka pierwszego jest podmiotem drugiego -> podrzędne przydawkowe: mianownik/dopełniacz, liczba pojedyncza/mnoga,

rodzaj męski/żeński/nijaki/męskoosobowy/niemęskoosobowy (oddzielne wzorce dla każdej kombinacji).

### Testy

Przeprowadzono testy poprawności podziału zdań złożonych na wybranych plikach rzeczywistych przetworzonych na poprzednich etapach analizy. Po przetworzenia danych przez program dla każdego pliku manualnie określono:

- liczba zdań złożonych,
- liczba zdań, które nadają się do podzielenia,
    - a wśród nich liczba zdań poprawnie opisanych (w celu eliminacji błędów popełnionych na wcześniejszych etapach analizy)
- liczba zdań podzielonych przez program,
    - a wśród nich liczba zdań podzielonych poprawnie (zachowujących poprawność gramatyczną i znaczeniową).
- poprawność potencjalna, liczona jako stosunek zdań poprawnie podzielonych do liczby zdań nadających się do podzielenia i jednocześnie poprawnie opisanych w danych wejściowych. Miara ta określa jaką poprawność uzyskałby program, gdyby zapewnić stuprocentową poprawność danych wejściowych.
- poprawność efektywna, liczona jako stosunek liczby zdań poprawnie podzielonych do liczby zdań nadających się do podzielenia (włącznie z błędnie lub niepełnie opisanymi). Miara ta określa poprawność całego procesu obejmującego zarówno zadanie podziału zdań złożonych jak i wcześniejsze etapy analizy.

Przedstawione są również czasy przetwarzania na komputerze z procesorem AMD 1,6GHz i 512MB RAM.

Wyniki przedstawia poniższa tabela.

| **Dane testowe** | **Miary poprawności** | **Czas przetwarzania [s]** |
|---|---|---|
| lalka1.txt | Zdań złożonych: 10<br>Do podzielenia: 9<br>W tym poprawnie opisanych: 2<br>Podzielonych: 2<br>Podzielonych poprawnie: 2<br>**Poprawność potencjalna: 100%**<br>**Poprawność efektywna: 22%** | 2 |
| MAGNETYZER.txt | Zdań złożonych: 10<br>Do podzielenia: 7<br>W tym poprawnie opisanych: 4<br>Podzielonych: 2<br>Podzielonych poprawnie: 2<br>**Poprawność potencjalna: 50%**<br>**Poprawność efektywna: 29%** | 2 |
| Naglowki.txt | Zdań złożonych: 36<br>Do podzielenia: 33<br>W tym poprawnie opisanych: 8<br>Podzielonych: 8<br>Podzielonych poprawnie: 5<br>**Poprawność potencjalna: 63%**<br>**Poprawność efektywna: 15%** | 3 |
| zdania.txt | Zdań złożonych: 6<br>Do podzielenia: 6<br>W tym poprawnie opisanych: 6<br>Podzielonych: 6<br>Podzielonych poprawnie: 6<br>**Poprawność potencjalna: 100%**<br>**Poprawność efektywna: 100%** | 2 |

Poniżej znajduje się pełniejsze omówienie konkretnych plików.

- *Lalka.txt*

  Zdania, które nie zostały podzielone, a powinny, najczęściej miały nierozpoznane czasowniki przez jeden z poprzednich programów. Wiele było

też zdań wtrąconych, które nie były obsługiwane. Pozostałe zostały poprawnie podzielone.

- *Magnetyzer.txt*

  W tym tekście niektóre zdania nie zostały podzielone z powodu błędnego oznaczenia czasowników w pliku wejściowym. Dwa zdania nie zostały podzielone z powodu błędu interpunkcyjnego: brak przecinka.

- *Naglowki.txt*

  W tym tekście słowo "może" nie zostało poprawnie opisane w danych wejściowych (część mowy nie została określona), a ponieważ wiele zdań złożonych występujących w tym tekście zawierało właśnie to słowo jako orzeczenie, program nie podzielił wielu z nich. W dwóch przypadkach, podczas dzielenia zdania podrzędnie złożonego przydawkowego wystąpienie tokenu "CRLF" przed słowem "który" uniemożliwiło wstawienie odpowiedniego podmiotu do drugiego zdania. Natomiast przy dzieleniu zdania współrzędnie złożonego łącznie z jednym podmiotem nie skopiowano podmiotu z pierwszego do drugiego zdania z powodu błędnego rozpoznawania pewnego rzeczownika w drugim ze zdań jako podmiotu.

- *zdania.txt*

  Plik zawiera zdania złożone przygotowane specjalnie do testowania tego zadania. Zawiera również zdania wielokrotnie złożone, które są poprawnie dzielone przez program. Bardziej wyrafinowana procedura podziału dotyczy np. zdań współrzędnie złożonych z jednym podmiotem, które są dzielone w taki sposób, że w obu zdaniach wynikowych znajdzie się podmiot. Zdania przydawkowe również dzielone są w taki sposób, że słowo określone przez zdanie podrzędne zostanie umieszczone w zdaniu podrzędnym po podziale.

Podsumowując, poprawność potencjalna jest zadowalająca i waha się w granicach 50%-100%. Jednak z powodu różnego rodzaju błędów, które wystąpiły na wcześniejszych etapach przetwarzania, poprawność efektywna dla plików rzeczywistych jest niska i waha się w granicach 15%-29%. Kluczowe znaczenie dla zadania ma poprawne opisanie tokenów w danych wejściowych. Tekst literacki dzieli się trudno ze względu na mnogość zdań złożonych będących wtrąceniami. Błędy interpunkcyjne w postaci

braku przecinka uniemożliwiają poprawny podział zdania, jednak nie zdarzają się często.

Poprawienie jakości danych wejściowych, czyli eliminacja błędów na poprzednich etapach analizy, może znacząco zwiększyć poprawność podziału zdań złożonych. Również rozszerzenie zbioru reguł i objęcie większego zbioru zdań złożonych może poprawić uzyskiwaną poprawność. W celu znaczącego poprawienia skuteczności programu dalsza linia rozwoju powinna skupiać się na poprawieniu jakości danych wejściowych i opracowaniu metody podziału zdań złożonych będących wtrąceniami.

Dodatkowo przetestowano możliwości łączenia zdań prostych w zdania złożone. Okazało się jednak, że testowane dokumenty w większości wypadków nie zawierały zdań, które mogą być połączone. W związku z tym przedstawione są wyniki jedynie dla plików *„Artshort.txt”*, który jest podzbiorem pliku *„Artykuły.txt”* oraz *„zdania.txt”*. W przypadku pierwszego pliku z powodu znacznej liczności zdań nie zostały określone wszystkie miary. Wyniki przedstawia poniższa tabela. W przypadku pozostałych testowanych plików nie zostały połączone żadne zdania.

| **Dane testowe** | **Miary poprawności** | **Czas przetwarzania [s]** |
|---|---|---|
| Artshort.txt | Zdań prostych: 1744<br>Połączonych: 23 | 51 |
| zdania.txt | Zdań prostych: 16<br>Do połączenia: 4<br>Połączonych: 4<br>Połączonych poprawnie: 4 | 1 |

Możliwości programu można zaobserwować na przygotowanym pliku *„zdania.xml”*. W pliku *„artshort.txt”*, w którym przy podziale powstało 108 nowych zdań, 23 zostały połączone przy przetwarzaniu odwrotnym. Wynika to przede wszystkim z faktu, że przygotowane reguły pokrywają jedynie ograniczony zbiór sytuacji, w których zdania mogą być łączone. Wykonane testy pokazują jednocześnie, że sytuacja, w której może nastąpić połączenie zdań prostych w złożone występuje stosunkowo rzadko w badanych dokumentach rzeczywistych. Jednak zagadnienie to może być pomocne w procesie

post-processing'u podczas tworzenia streszczeń. Wymaga to jednak przygotowania odpowiedniego specyficznego zbioru reguł łączących.

## 3.8. Rozbiór zdań

Kolejnym elementem analizy STP jest przeprowadzenie rozbioru zdania. W zależności od celu analizy oraz planowanego zastosowania analizy STP istnieje kilka możliwych podejść. Czasami potrzebne jest przeprowadzenie pełnego rozbioru zdań obejmującego wszystkie słowa (tokeny) występujące w tekście wraz z określeniem ich wzajemnych relacji. Takie podejście wywodzi się z analizy NLP. Natomiast w wielu zastosowaniach wystarczające okazuje się oznakowanie części zdania lub też rozpoznanie grup części mowy (np. grupa rzeczownika, grupa czasownika) lub też grup części zdania (np.: grupy podmiotu, grupy orzeczenia). Z tego powodu zadanie rozbioru zdań bywa dzielone hierarchicznie na dwa podzadania:

- **rozpoznawanie fragmentów / grup wyrazów** (ang. ***fragment recognition***),
- **rozpoznawanie klauzul** (ang. ***clause recognition***)

Rozpoznawanie grup wyrazów może być przeprowadzane w sposób uproszczony, bez pełnej analizy lingwistycznej. Może ona bazować na prostych regułach określających granice grupy wyrazów na podstawie rozpoznanych części mowy oraz nazw własnych. Proponowane były zarówno podejścia oparte na regułach [Abney, 1991], jak i korzystające z technik statystycznych [Ting, 1995].

Rozpoznawanie klauzul polega na rozpoznawaniu struktur większych niż grupy wyrazów, czyli tworzenie grup składających się z grup. Grupy takie mogą być rekurencyjnie zagnieżdżone [Sag, Wasow, 1999]. Często jest tutaj również uwzględniane rozpoznawanie zdań podrzędnych, w szczególności zdań wtrąconych.

W większości znanych systemów analizy STP stosowana jest strategia „z dołu do góry" (ang. ***bottom-up***) do rozpoznawania fragmentów oraz rekurencyjnych struktur zdania [Sundheim, 1995], [SAIC, 1998]. Taka strategia polega na grupowaniu prostych grup i słów w większe struktury syntaktyczne. Jeśli jednak zostanie popełniony błąd w rozpoznaniu na najniższym poziomie, to jest on propagowany i może zaburzyć lub nawet uniemożliwić rozpoznanie całego zdania. Lepsze rezultaty można uzyskać łącząc strategię „z dołu do góry" z pewnymi elementami analizy „z góry do dołu" (ang. ***top-***

***down***), np. na początku rozpoznając granice zdań podrzędnych i przeprowadzając dalszą analizę dla każdego z nich oddzielnie [Peh, Ting, 1996], [Piskorski, 2001].

Podobne rezultaty uzyskano dla języka polskiego w ramach prac badawczych prowadzonych przez autora niniejszej rozprawy. Aby przetestować i porównać różne sposoby rozbioru zdań opracowano kilka algorytmów i wykonano kilka implementacji. Poniżej znajduje się szersze omówienie zastosowanych podejść i uzyskanych wyników.

Dla przypomnienia krótki opis części zdania znajduje się w rozdziale 7. Załącznik B: Krótki opis części zdania w j. polskim (str. 145).

### 3.8.1. Podejście rekurencyjne

W celu przeprowadzeniu pełnego rozbioru zdań przy podejściu rekurencyjnym potrzebne jest przygotowanie zbioru hierarchicznych reguł definiujących możliwe przypadki rozbioru. Każda z reguł określa jeden z węzłów drzewa rozbioru. W danym węźle może oczywiście być wiele reguł określających różne warianty drzewa rozbioru. Reguły mają następującą postać BNF (Backus-Naur Form):

```
ZdanieZlozone ::= ZdaniePojedyncze
ZdanieZlozone ::= ZdaniePojedyncze Lacznik-Rownorzedny ZdaniePojedyncze
ZdanieZlozone ::= GrupaPodmiotu_ Lacznik-Wzgledny1 ZdaniePojedyncze
Lacznik-Wzgledny2 GrupaOrzeczenia
ZdaniePojedyncze ::= GrupaOrzeczenia
ZdaniePojedyncze ::= GrupaPodmiotu_ GrupaOrzeczenia
ZdaniePojedyncze ::= GrupaOrzeczenia GrupaPodmiotu_
GrupaPodmiotu ::= Podmiot
GrupaPodmiotu ::= GrupaPrzydawki Podmiot_
GrupaPodmiotu ::= Podmiot_ GrupaPrzydawki
```

Przetwarzanie odbywa się w konwencji zstępującej. Jako element główny występuje element *ZdanieZlozone*, które dzieli się na grupy, grupy dzielą się na grupy, itd. Rozbiór zdania oznacza więc budowanie drzewa na podstawie wielu jego wariantów dopuszczanych przez reguły. Reguła może się składać z od 1 do *n* produktów – podelementów reguły. Produkt może zawierać element terminalny (odpowiednią część mowy) lub odniesienie do innej reguły. Dodatkowo w regułach definiowane są związki zgodności, np. zgodność liczby i rodzaju przydawki z podmiotem.

W regułach ważna jest kolejność produktów, muszą one występować w podanej w regule kolejności, żeby ją spełniać. Jeżeli więc np. jakieś dwa produkty mogą

występować obok siebie w różnej kolejności, to konieczne jest wstawienie dwóch reguł. Możliwe jest także stosowanie „wewnętrznej rekurencji", czyli dany element (a ściślej grupa, nie element terminalny), może jako jeden z produktów zawierać sam siebie, np.:

*grupa_podmiotu ::= grupa_przydawki podmiot , grupa_podmiotu*

Pozwala to na stosowanie ciągów wystąpień np. grup przydawkowych, bez konieczności stosowania sztywnych reguł dla powtórzeń.

Wykonano dwie implementacje [Proj11], [Proj12]. Przy implementacji problemem jest stworzenie pełnego zbioru reguł rozbioru zdań. Wynika to ze swobodnego szyku zdań w języku polskim. Poziom komplikacji reguł dodatkowo wzrasta po uwzględnieniu zdań złożonych, w tym także zdań wtrąconych. Rozszerzanie zbioru reguł o kolejne możliwe sytuacje powoduje znaczne rozszerzanie drzewa poszukiwań i wydłuża czas analizy.

Testowane implementacje bardzo dobrze radzą sobie z rozkładem zdań krótkich (tj. do ok. pięciu tokenów). W przypadku zdań dłuższych drzewo przeszukiwań staje się na tyle mocno rozpięte, że przeszukanie go zajmuje zbyt wiele czasu. Można podejmować próby zoptymalizowania algorytmu, aczkolwiek żaden zabieg nie poprawi radykalnie jego efektywności czasowej. Język polski jest na tyle skomplikowany, że drzewo możliwych rozbiorów zdań będzie zawsze bardzo rozbudowane. Wynika to z faktu, że w języku polskim szyk słów w zdaniu ma minimalne znaczenie. Istnieje wiele zdań, w których można dowolnie poprzestawiać słowa, a zawsze będą one niosły ze sobą tę samą treść. Z tego względu, podejście rekurencyjne nadaje się jedynie do rozbioru krótkich zdań lub części zdań (po uprzedniej dekompozycji zdania na mniejsze, spójne logicznie części).

### 3.8.2. Podejście statystyczno-adaptacyjne

W ramach prac badawczych przetestowano również podejście statystyczno-adaptacyjne [Proj13]. Przyjęto tutaj kilka założeń upraszczających zgodnie z ideologią płytkiej analizy (STP):

- Płaska struktura analizy – tokenom przyporządkowywana jest ich rola w zdaniu, np. podmiot, jednak bez pełnej analizy zależności pomiędzy tokenami. W szczególności nie jest budowana hierarchiczna struktura drzewiasta.

- Lokalność analizy – analiza obejmuje wyłącznie pewne sąsiedztwo tokenu. Proces rozpoznawania części zdania sprowadzono do zadania klasyfikacji na podstawie atrybutów analizowanego tokenu i jego sąsiadów.

W związku z wykorzystywaniem klasyfikacji konieczne jest przeprowadzenie etapu uczenia klasyfikatora. Ogólnie uczenie oparte jest na budowaniu wzorców na podstawie wartości atrybutów tokenu, który służy jako przykład oraz na podstawie wartości atrybutów jego sąsiadów. Rozmiar sąsiedztwa może być ustalany oraz mogą być wybrane atrybuty, które będą klasyfikowane. Podczas klasyfikacji, dla danego tokena przeglądana jest cała lista reguł i wybierane są te, które są najlepiej dopasowane do tokena.

Wyniki testów w dużym stopniu zależą od obszerności i różnorodności danych trenujących. Jednak zastosowana metoda nie wymaga od operatora znajomości wszystkich szczegółowych reguł składni języka polskiego. Takie podejście pozwala na łatwiejsze dostosowywanie rozwiązania do nowych dziedzin i rodzajów tekstów. W szczególności może być ono bez większych modyfikacji zastosowane do analizy innych języków. Z drugiej strony wymaga to przygotowania odpowiednio dużego opisanego zbioru trenującego. Wyraźną wadą takiego podejścia jest również uzyskiwanie płaskiej struktury wyników. Jednym z możliwych rozszerzeń mogłoby być stworzenie klasyfikatora, który uczyłby się również zależności hierarchicznych w celu tworzenia drzewiastej struktury wyników.

#### Przykład działania

Token przykładowy:

*(id = 5, przyimek; dopełnienie)*

Tokeny sąsiednie:

*(id = 4, czasownik, czas przeszły) (id = 4, czasownik, strona bierna)*
*(id = 6, rzeczownik, biernik) (id = 6, rzeczownik, mianownik)*

będą podstawą do wygenerowania następujących reguł:

*(id = 4, czasownik, czas przeszły) (id = 5, przyimek; dopełnienie) (id = 6, rzeczownik, mianownik)*

*(id = 4, czasownik, czas przeszły) (id = 5, przyimek; dopełnienie) (id = 6, rzeczownik, biernik)*

*(id = 4, czasownik, strona bierna) (id = 5, przyimek; dopełnienie) (id = 6, rzeczownik, mianownik)*

*(id = 4, czasownik, strona bierna) (id = 5, przyimek; dopełnienie) (id = 6, rzeczownik, biernik)*

Wygenerowane reguły poddane są redukcji. W procesie tym za lepsze, uznawane są reguły bardziej szczegółowe, czyli posiadające więcej atrybutów opisujących otoczenie klasyfikowanego tokenu. Wybranie reguł bardziej szczegółowych jako lepszych od bardziej ogólnych daje lepsze własności klasyfikowania elementów złożonych struktur. Przykładowo dane jest zdanie:

*Gołąb na parapecie spogląda do mieszkania.*

Przypuśćmy, że w procesie uczenia wyłoniono następujące reguły:

*(-) (przyimek; przydawka) (-)*
*(-) (przyimek; dopełnienie) (-)*
*(czasownik) (przyimek; dopełnienie) (rzeczownik)*
*(-) (rzeczownik; przydawka) (-)*
*(-) (rzeczownik; dopełnienie) (-)*
*(przyimek) (rzeczownik; dopełnienie) (-)*

Wówczas przy strategii preferowania reguł bardziej ogólnych, po redukcji zostaną uzyskane reguły:

*(-) (przyimek; przydawka) (-)*
*(-) (przyimek; dopełnienie) (-)*
*(-) (rzeczownik; przydawka) (-)*
*(-) (rzeczownik; dopełnienie) (-)*

W takim przypadku oba przyimki w zdaniu (‘na’, ‘do’) nie będą zaklasyfikowane jednoznacznie – uzyskane zostaną wynikowe tokeny:

*(id=1 slowo=’na’ czescmowy=’przyimek’ czesczdania=’przydawka’)*
*(id=1 slowo =’na’ czescmowy=’przyimek’ czesczdania=’dopelnienie’)*
*(id=4 slowo =’do’ czescmowy=’przyimek’ czesczdania=’przydawka’)*
*(id=4 slowo =’do’ czescmowy=’przyimek’ czesczdania=’dopelnienie’)*

Podobnie z rzeczownikami ‘parapecie’ oraz ‘mieszkania’. W przypadku preferowania reguł bardzie szczegółowych, odrzucone zostaną reguły:

*(-) (przyimek; dopełnienie) (-)*
*(-) (rzeczownik; dopełnienie) (-)*

a pozostaną:

*(-) (przyimek; przydawka) (-)*
*(czasownik) (przyimek; dopełnienie) (rzeczownik)*
*(-) (rzeczownik; przydawka) (-)*
*(przyimek) (rzeczownik; dopełnienie) (-)*

wtedy klasyfikacja będzie o wiele celniejsza, a mianowicie:

- Dla rzeczownika ‘parapecie’ reguła 3 będzie lepiej dopasowana niż reguła 4, ponieważ z lewej strony tego wyrazu nie ma przyimka, stąd: ‘parapecie’ -> przydawka
- Dla rzeczownika ‘mieszkania’ lepiej dopasowana będzie reguła 4, ponieważ z jego lewej strony jest przyimek

Jak wiadomo, w różnych algorytmach wykorzystujących reguły asocjacyjne również preferuje się reguły dłuższe. Mają one silniejsze własności klasyfikacyjne oraz są nadzbiorami reguł bardziej ogólnych.

### 3.8.3. Podejście relacyjne

W ramach prac badawczych opracowano również podejście relacyjne [Proj14], które jest pewnym kompromisem pomiędzy złożonością i precyzją w stosunku do rozbioru rekurencyjnego. Podejście to jest w pewnym stopniu podobne do podejścia statystycznego, jednak zamiast klasyfikatora nauczonego na przykładach stosowany jest zbiór reguł relacyjnych, które określają relacje między częściami zdania. Przy czym aplikowane w określonym porządku reguły odnoszą się kolejno do różnych części zdania. Taka metodologia przypomina sposób działania człowieka, który działając w obrębie zdania jest w stanie najpierw zaznaczyć orzeczenie, następnie podmiot, określające go przydawki itd. Reguły relacyjne, wspierając się występowaniem znaków przystankowych, są w stanie również rozpoznać i oddzielić zdania złożone, w tym także zdania wtrącone. Reguły te pozwalają również na znalezienie pewnych zależności pomiędzy wyrazami (tokenami). Nie jest tutaj budowana pełna hierarchia drzewiasta,

jak w przypadku analizy rekurencyjnej. Jednak w praktycznych zastosowaniach nie wszystkie zależności są potrzebne.

Wadą takiego rozwiązania jest duży poziom skomplikowania reguł i trudność ich syntetycznego zapisu o mniejszej sile wyrazu niż występującej we współczesnych językach programowania. Utrudnia to tworzenie i modyfikację takich reguł. Stąd dopasowanie analizy do zmieniających się warunków (inna dziedzina, ewolucja języka, slang itp.) jest dosyć skomplikowane. Z drugiej jednak strony analiza nie jest pełna i umożliwia w większości wypadków pomijanie „nieznanych" konstrukcji i sformułowań bez uszczerbku dla „rozpoznawanej" części tekstu.

Jednak ze względu na zadowalające wyniki wykonano nowa implementacja [Proj15], w której do opisu reguł określania części zdania użyto zewnętrzny słownik. Pozwoliło to na uzyskanie większej swobody definiowania i modyfikacji reguł. Reguły określają zależności, jakie powinny spełnić dwa tokeny, żeby można było powiedzieć, że jeden z nich określa drugi, i w jaki sposób (tzn. jaką jest częścią zdania).

Algorytm rozpoczyna działanie od wyodrębnienia poszczególnych podzdań całego zdania (o ile jest złożone). Następnie rozbiór jest dokonywany zgodnie ze strategią ***top-down***. Algorytm znajduje orzeczenie w zdaniu oraz ewentualny podmiot (na podstawie reguł – w tym przypadku nie szuka związku wyrazów, tylko wyrazu odpowiadającego kryteriom). Następnie zaś przeszukuje te wyrazy danego podzdania, dla których nie znaleziono jeszcze określenia, próbując znaleźć wyraz, który określają. Przeszukiwanie to szuka powiązań z wyrazami, dla których znaleziono rozwiązanie w poprzednim kroku, co ułatwia hierarchiczne konstruowanie wyniku.

Program korzysta z dwóch plików reguł: *„regulygr.xml"* oraz *„reguly.xml"*. Pierwszy z nich definiuje sposób podziału zdań na podzdania, drugi zawiera reguły konstrukcji związków wyrazowych.

Reguły podziału na podzdania definiowane są następująco: element *regulagr* zawiera dwa atrybuty: *wynik* oraz *separator*. Wynik zawiera określenie zdania składowego (podrzędne, współrzędne), separator zawiera znak, jaki oddziela zdania od siebie (np. „„", „„."). Ponadto każdy element *regulagr* może zawierać dowolną ilość warunków dodatkowych (*warunekgr*), jakie musi spełniać słowo następujące po separatorze. Warunki interpretowane są jako połączone logiczną koniunkcją – a więc muszą zostać

spełnione wszystkie, aby spełniona została reguła. Pojedynczy warunek posiada dwa atrybuty (*lvalue, rvalue*), które dyktują, jakie pole danego tokena zostanie sprawdzone. Możliwe wartości to „*slowo*”, „*rodzajTokena*”, „*lemat*”, „*czescMowy*”, „*przypadek*”, „*typ*”, „*strona*”, „*licznosc*”, „*tryb*”, „*rodzaj*”, „*stopien*”, „*czas*”, bądź też inny, dowolny napis. Jeśli podana wartość nie zawiera się w podanym zbiorze, to nie będzie ona determinować wyboru pola tokenu, tylko będzie wartością do porównania (np. można zadeklarować, że przypadek ma być mianownikiem – warunek będzie wyglądał: `<warunekgr lvalue=”mianownik” rvalue=”slowo”></warunek>`. Oznaczenie konkretnego pola musi być wartością atrybutu *rvalue*. Przykładem takiej reguły jest poniższa:

```
<regulagr wynik="zdanie podrzedne" rodzaj="zdanie" separator=",">
            <warunekgr lvalue="że" rvalue="slowo"></warunekgr>
</regulagr>
```

Reguła ta stwierdza, iż zdanie składowe, oddzielone od poprzedniego przecinkiem, i takie, którego pierwszym słowem jest słowo „który”, jest zdaniem podrzędnym w stosunku do poprzedniego.

Reguły określające związki międzywyrazowe są zbudowane na podobnej zasadzie, z tą różnicą, że oznaczenia pól mogą występować po obu stronach, i dotyczą dwóch tokenów. Typowa reguła zapisana w pliku *reguly.xml* zawiera atrybut wynik, który określa część zdania, jaką reprezentuje wyraz określający (jeśli spełni daną regułę), oraz dowolną ilość warunków (zapisanych jako podelementy), jakie muszą spełniać wyrazy, by zachodził między nimi określany regułą związek. Każdy warunek posiada dwa atrybuty: *lvalue* i *rvalue*. Atrybuty *rvalue* dotyczą wyrazu określanego, atrybuty *lvalue* – określającego. Możliwymi wartościami są: „slowo”, „rodzajTokena”, „lemat”, „czescMowy”, „przypadek”, „typ”, „strona”, „licznosc”, „tryb”, „rodzaj”, stopien”, „czescZdania”, „czas” (jeśli mają odnosić się do pola tokena), bądź dowolny inny napis (traktowany jako bezpośrednio podana wartość). Aby została spełniona reguła, muszą zostać spełnione wszystkie jej warunki. Przykładem takiej reguły jest:

```
<regula wynik="przydawka">
            <warunek lvalue="czescMowy" rvalue="przymiotnik"></warunek>
            <warunek lvalue="podmiot" rvalue="czescZdania"></warunek>
```

```
            <warunek lvalue="przypadek" rvalue="przypadek"></warunek>
</regula>
```

Ta reguła określa, iż przymiotnik o tym samym przypadku, co podmiot, określa go, a tym samym jest częścią zdania zwaną przydawką.

### Testy

Niestety w czasie testów okazało się, że w większości wypadków nie udaje się dokonać pełnego rozbioru, chociaż udaje się dokonać rozbioru częściowego, czyli określenia niektórych części zdania i relacji między nimi. Okazało się, że w dużym stopniu wynika to z różnego rodzaju braków występujących w danych wejściowych. Rozbiór pełny nie udawał się z powodu niedostatecznego określenia tokenów (głównie części mowy, i ich specyfikacji – przypadków, rodzajów, itp.). Braki te są spowodowane niedoskonałościami narzędzi wykonujących poprzednie etapy płytkiej analizy tekstu, gdyż błędy te się kumulują na kolejnych etapach. W celu sprawdzenia potencjalnej skuteczności zastosowanej metody i przygotowanych reguł wybrane początkowe fragmenty rzeczywistych dokumentów sprawdzono i poprawiono manualnie. Poprawione zdania zostały poddane automatycznemu rozbiorowi. Uzyskany w ten sposób wynik opisany jest jako skuteczność potencjalna.

Podane czasy wykonania obejmują przetwarzanie całych plików (nie tylko testowanych fragmentów) i były liczone na komputerze z procesorem AMD 1,6GHz.

| **Dane testowe** | **Miary poprawności** | **Czas przetwarzania całego dokumentu** |
|---|---|---|
| Bankowa superfuzja - UniCredito przejmuje HVB.txt | Liczba wszystkich zdań: 76<br>Liczba testowanych: 18<br>Liczba zdań poprawnie rozłożonych: 17<br>Liczba zdań częściowo poprawnie rozłożonych: 1<br>**Efektywna skuteczność: 22%**<br>**Potencjalna skuteczność: 94%** | 10s |
| Młodzi w Polsce przestaja być bierni i bezradni .txt | Liczba wszystkich zdań: 37<br>Liczba testowanych: 13<br>Liczba zdań poprawnie rozłożonych: 11<br>Liczba zdań częściowo poprawnie rozłożonych: 2<br>**Efektywna skuteczność: 30%**<br>**Potencjalna skuteczność: 85%** | 1,4s |
| Pierwsze dziecko z przeszczepu jajnika innej kobiety.txt | Liczba wszystkich zdań: 43<br>Liczba testowanych: 15<br>Liczba zdań poprawnie rozłożonych: 13<br>Liczba zdań częściowo poprawnie rozłożonych: 2<br>**Efektywna skuteczność: 30%**<br>**Potencjalna skuteczność: 87%** | 2,6s |

Jak widać w powyższej tabeli potencjalna skuteczność jest zadowalająca. Zdecydowana większość zdań została całkowicie rozłożona w sposób poprawny, a pozostałe zdania zostały rozłożone częściowo. Niestety błędy pojawiające się w danych testowych

będących bezpośrednim wynikiem poprzednich etapów płytkiej analizy tekstu w znaczący sposób zmniejszają tą skuteczność. Wynika to w dużym stopniu z szeregowego przetwarzania kolejnych etapów analizy i ich wzajemnych zależności. Błędy występujące na jednym etapie mogą powodować błędy na kolejnych etapach. Z tego powodu błędy ulegają kumulacji.

Podsumowując, zastosowane podejście do rozbioru zdań jest słuszne i może być skuteczne pod warunkiem minimalizacji ilości błędów na wcześniejszych etapach analizy.

# 4. Zadania TDM wspierane przez techniki płytkiej analizy tekstu

W literaturze przedstawiono wiele algorytmów realizujących zadania TDM. Najpopularniejsze algorytmy oraz reprezentacje tekstu przedstawiono w rozdziale 2. „Techniki eksploracji tekstu" (str. 19). W omówionych algorytmach i technikach przekształcania reprezentacji intensywnie wykorzystywane jest podejście statystyczne, co jest typowe dla technik eksploracji tekstu. W niniejszym rozdziale omówiono możliwości wykorzystania dodatkowych informacji lingwistycznych uzyskiwanych w procesie płytkiej analizy tekstu. Podstawowym celem jest sprawdzenie możliwości poprawy wyników poprzez rozszerzenie reprezentacji oraz dostosowanie algorytmów. W kolejnych podrozdziałach przedstawiono wyniki uzyskane dla zadań klasyfikacji i grupowania. Omówiono również zadanie sumaryzacji dokumentów.

## *4.1. Klasyfikacja dokumentów wspierana przez techniki płytkiej analizy tekstu*

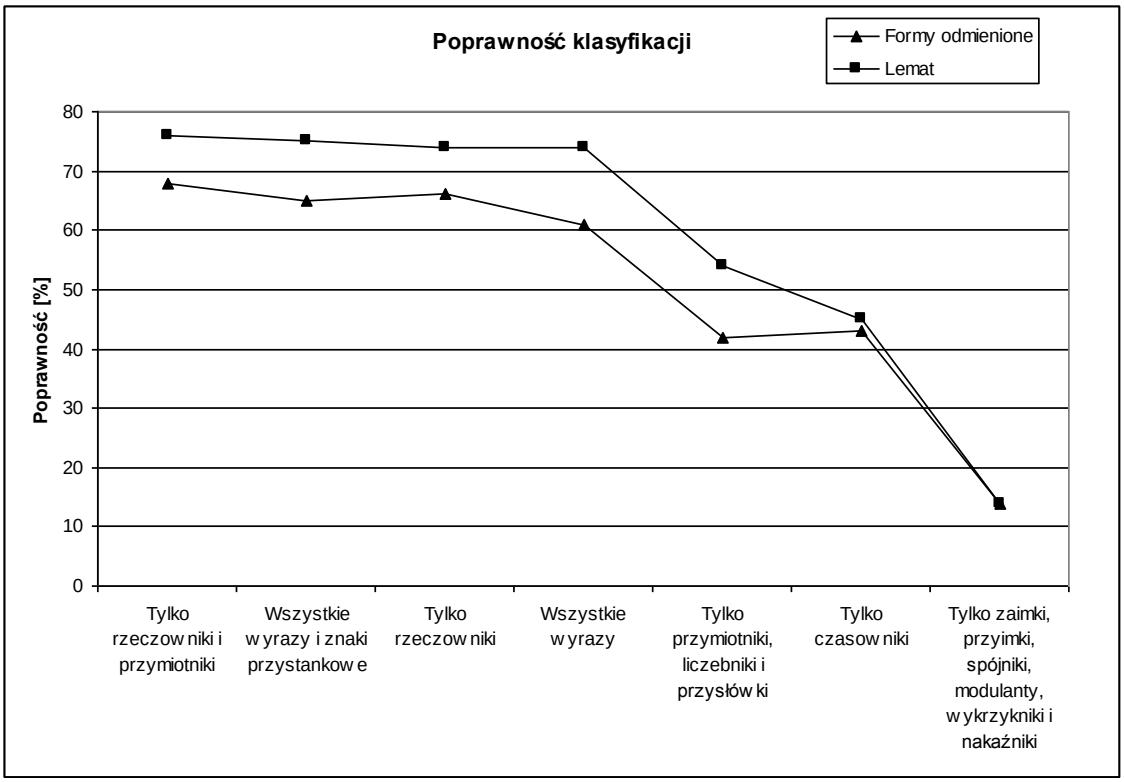
Jednym z zadań eksploracji tekstu jest klasyfikacja dokumentów. Zadanie to przedstawiono w rozdziale 1.3.1 „Zadania TDM" (str. 11). Wybrane algorytmy realizacji tego zadania omówiono w rozdziale 2.3. „Algorytmy klasyfikacji dokumentów" (str. 23). W niniejszym rozdziale zaproponowano możliwości wykorzystania technik płytkiej analizy tekstu do poprawy wyników klasyfikacji dokumentów. Przedstawiono również wyniki testów na wybranych algorytmach.

Typowe podejście stosowane w algorytmach klasyfikacji dokumentów polega na przygotowaniu odpowiedniej reprezentacji dokumentów, zwykle wielowymiarowych wektorów, a następnie zastosowaniu wybranego algorytmu grupowania. Reprezentacja wektorowa jest zwykle ograniczana poprzez techniki ***pruning***'u (usuwanie wybranych słów z reprezentacji). Jednak wykorzystywane są do tego statyczne listy słów, które muszą być ręcznie dopasowywane do dziedziny. Dzięki wykorzystaniu technik płytkiej analizy tekstu możliwe jest przekształcanie reprezentacji z wykorzystaniem dodatkowych informacji lingwistycznych.

Aby przetestować wpływ zastosowania tej techniki przeprowadzono eksperyment, w ramach którego sprawdzono wpływ usuwania z reprezentacji wybranych części mowy. Nie przetestowano wpływ innych informacji lingwistycznych, takich jak forma gramatyczna czy informacja o funkcji wyrazu w zdaniu. Wynika to z faktu, że stworzone narzędzia określają informację o częściach mowy z dużym poziomem poprawności, a uzyskane wyniki powinny być najłatwiejsze do interpretacji. Interpretacja odrzucania wybranych form gramatycznych wydaje się mniej intuicyjna. Natomiast poprawność określenia części zdania jest niższa niż części mowy. Do testów wybrano jeden z najpopularniejszych algorytmów klasyfikacji: Rocchio i kosinusowa miara podobieństwa wektorów. Wykorzystano reprezentację unigramowa i ważenie TF-IDF. Wybór ten jest arbitralny, jednak uzyskane wyniki są możliwe do uogólnienia. Testy przeprowadzono na zbiorze danych „Onet", który został omówiony w Załącznik D: Opis danych testowych (str.152). W przeprowadzonym eksperymencie dane trenujące i testowe były rozłączne, aby warunki eksperymentu były bardziej realistyczne. W przypadku klasyfikacji zbioru, który był użyty do uczenia klasyfikatora, poprawność klasyfikacji wynosi ponad 95%, co potwierdza trafność wyboru algorytmu i miary.

Wyniki przedstawia poniższa tabela.

| **Algorytm Rocchio<br>Miara kosinusowa<br>Dane testowe** | **Miara poprawności klasyfikacji [%]** | |
|---|---|---|
| | **Formy odmienione** | **Lemat** |
| Wszystkie wyrazy i znaki przystankowe | 65 | 75 |
| Wszystkie wyrazy | 61 | 74 |
| Tylko rzeczowniki | 66 | 74 |
| Tylko rzeczowniki i przymiotniki | 68 | **76** |
| Tylko czasowniki | 43 | 45 |
| Tylko przymiotniki, liczebniki i przysłówki | 42 | 54 |
| Tylko zaimki, przyimki, spójniki, modulanty, wykrzykniki i nakaźniki | 14 | 14 |
| Wszystkie wyrazy oprócz rzeczowników | 42 | 63 |
| Wszystkie wyrazy oprócz rzeczowników i przymiotników | 44 | 44 |




Na podstawie zaprezentowanych powyżej wyników można wysnuć kilka wniosków. Po pierwsze wykorzystanie form podstawowych wyrazów (lematów) pozwala wyraźnie poprawić poprawność klasyfikacji. W podanym przykładzie poprawność wzrasta o około 10%. Jest to w pełni zgodne z intuicją, gdyż w kontekście klasyfikacji wartość znaczeniowa wyrazu nie jest zależna od odmiany.

Na podstawie uzyskanych wyników można również wnioskować, że najważniejsza z punktu widzenia klasyfikacji informacja jest przechowywana w rzeczownikach. Duże znaczenie mają również przymiotniki, zwłaszcza w połączeniu z rzeczownikami. Pozostałe części mowy są wyraźnie mniej znaczące i mogą nawet obniżać poprawność klasyfikacji poprzez wprowadzanie szumu informacyjnego. Wobec tego najlepsze efekty klasyfikacji można uzyskać stosując wyłącznie słowa kluczowe w postaci rzeczowników i przymiotników w formie podstawowej. Taki wniosek jest również zgodny z intuicją, chociaż dla pewnych dziedzin danych może nie być prawdziwy. Można sobie wyobrazić specyficzny zbiór danych, w którym klasy mogą być odróżniane wyłącznie w oparciu o inne części mowy. Jednak w większości danych rzeczywistych stosowanie rzeczowników i przymiotników pozwala najlepiej reprezentować ich tematykę, przez co przeprowadzać skuteczną klasyfikację.

Podsumowując, wykorzystanie analizy morfologicznej pozwala poprawić wyniki klasyfikacji dokumentów tekstowych. Można również oczekiwać, że w konkretnych zastosowaniach wykorzystanie kolejnych etapów analizy STP może dawać dalszą poprawę wyników.

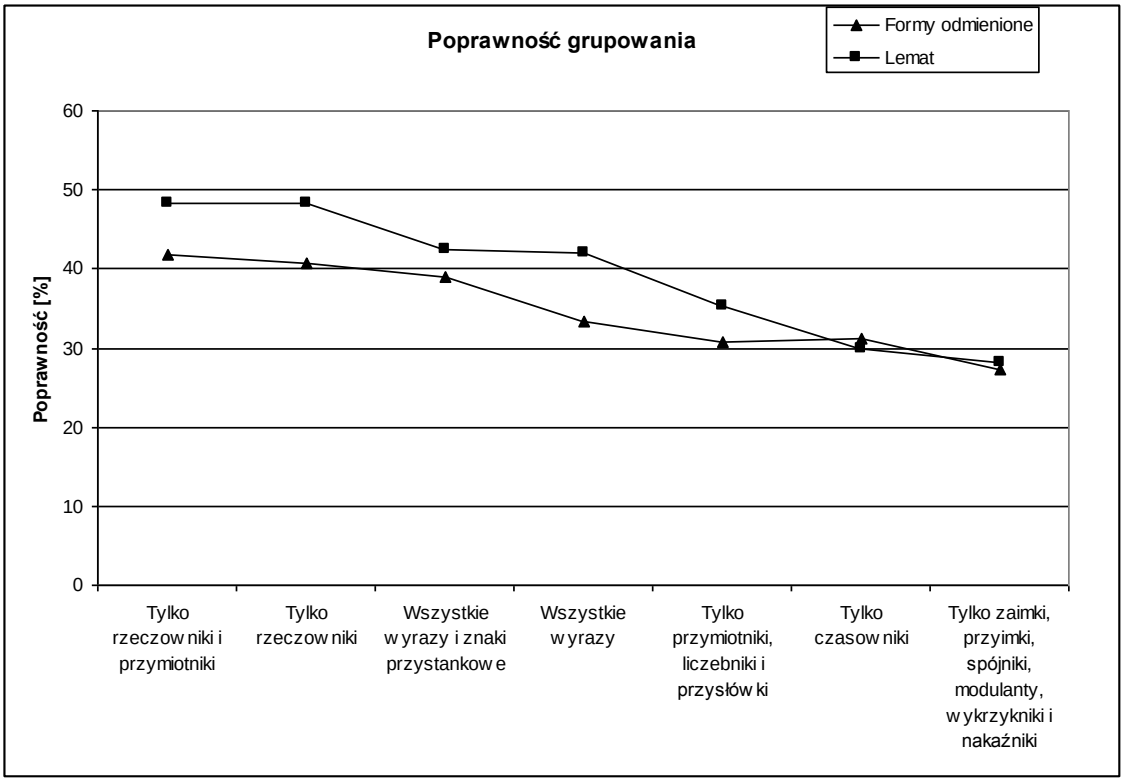
## 4.2. Grupowanie dokumentów wspierane przez techniki płytkiej analizy tekstu

Kolejnym zadaniem eksploracji tekstu, które przetestowano w ramach niniejszej rozprawy jest grupowanie dokumentów. Zadanie przedstawiono w rozdziale 1.3.1 „Zadania TDM" (str. 11). Wybrane algorytmy realizacji tego zadania omówiono w rozdziale 2.4 „Algorytmy grupowania dokumentów" (str. 26). W niniejszym rozdziale zaproponowano wykorzystanie technik płytkiej analizy tekstu do poprawy wyników grupowania dokumentów na przykładzie algorytmu k-Means z zastosowaniem miary kosinusowej, reprezentacji unigramowej z ważeniem TF-IDF. Algorytm ten jest jednym z najpopularniejszych algorytmów, który daje stosunkowo dobre wyniki. Wykorzystanie dodatkowych informacji lingwistycznych będących wynikiem analizy STP wpływa głównie na reprezentację tekstu i jest w dużym stopniu niezależne od wybranego algorytmu grupowania. W ramach eksperymentu przeprowadzono testy wykorzystania wyników analizy morfologicznej.

W algorytmie ustalono wymaganą liczbę klastrów równą 10, czyli zgodną z liczbą klas w danych źródłowych. Sposób obliczania poprawności opisano w rozdziale 2.6.2. Poprawność grup (str. 37).

Przeprowadzone testy oraz ich wyniki przedstawia poniższa tabela. Ze względu na to, iż w algorytmie k-Means występuje element niedeterministyczny (losowanie punktów startowych), wyniki mogą się w niewielkim zakresie zmieniać. Aby zminimalizować wpływ elementu losowego na wyniki każdy z testów został przeprowadzony trzykrotnie, a w tabeli umieszczono średni wyniki z trzech pomiarów.

| **Algorytm k-Means<br>Miara kosinusowa<br>Dane testowe** | **Miara poprawności grupowania [%]** | |
|---|---|---|
| | **Formy odmienione** | **Lemat** |
| Wszystkie wyrazy i znaki przystankowe | 39,0 | 42,4 |
| Wszystkie wyrazy | 33,3 | 42,1 |
| Tylko rzeczowniki | 40,8 | 48,3 |
| Tylko rzeczowniki i przymiotniki | 41,9 | **48,4** |
| Tylko czasowniki | 31,1 | 29,9 |
| Tylko przymiotniki, liczebniki i przysłówki | 30,8 | 35,4 |
| Tylko zaimki, przyimki, spójniki, modulanty, wykrzykniki i nakaźniki | 27,3 | 28,2 |
| Wszystkie wyrazy oprócz rzeczowników | 31,8 | 32,8 |



Analiza powyższych wyników prowadzi do podobnych wniosków jak w przypadku klasyfikacji. Po pierwsze wykorzystanie form podstawowych zamiast odmienionych daje dobre rezultaty. Po drugie wykorzystanie wyłącznie rzeczowników lub rzeczowników i przymiotników łącznie wyraźnie poprawia poprawność grupowania. Należy zwrócić uwagę, że zastosowana miara poprawności grupowania jest związana z wzorcowym przyporządkowaniem dokumentów do klas, które zostało wykonane przez człowieka. Tak więc miara uwzględnia zgodność podziału dokumentów z ludzkim pojmowaniem tematyki dokumentów, co okazuje się być związane głównie z występującymi w tekście rzeczownikami. Okazuje się, że są one bardziej „specyficzne" dla podziału klas niż inne części mowy. Co ciekawe uwzględnianie pozostałych części mowy może zakłócać proces grupowania i obniżać poprawność. Dzięki wykorzystaniu dodatkowych informacji lingwistycznych możliwe jest poprawianie jakości grupowania dokumentów tekstowych.

## 4.3. Sumaryzacja wspierana przez techniki STP

Porównując ekstrakcyjne i generacyjne podejście do sumaryzacji można zauważyć analogię do postulatów płytkiej i głębokiej analizy tekstu. Podejście generacyjne bardziej pasuje do postulatów głębokiej analizy tekstu, gdyż wiąże się z obszerniejszą analizą lingwistyczną oraz bardziej złożonymi algorytmami, co ma prowadzić do pełniejszego „zrozumienia" tekstu. Natomiast podejście ekstrakcyjne bliższe jest postulatom płytkiej analizy tekstu. Stosowany jest ograniczony zakres analiz lingwistycznych, często bazujący na metodach statystycznych. Powoduje to jednocześnie skrócenie czasu przetwarzania.

Mając na względzie powyższe powody w niniejszej rozprawie zaproponowano wykorzystanie metod płytkiej analizy tekstu do wzbogacenia ekstrakcyjnych technik sumaryzacji dokumentów tekstowych. Przedstawiono rozważania dotyczące języka polskiego, chociaż można je uogólnić również na inne języki fleksyjne. Zaproponowane rozwiązania sprawdzono również eksperymentalnie.

Poniższa tabela przedstawia problemy występujące w podczas stosowania ekstrakcyjnych metod sumaryzacji oraz sposoby ich rozwiązywania lub łagodzenia ich skutków w oparciu o techniki STP opisane w rozdziale 3. Model płytkiej analizy tekstu (str. 50).

| Problem | Sposób rozwiązania |
|---|---|
| Fleksja wyrazów powoduje, że różne formy tego samego wyrazu są traktowane jako inne wyrazy pomimo identycznej wartości semantycznej. | Wykorzystanie analizy morfologicznej w celu uzyskania form podstawowych wyrazów (lematów), które są wykorzystywane w dalszym przetwarzaniu, w szczególności do porównywania wyrazów i zdań. |
| Nazwy własne mogą być pomijane w streszczeniu, gdyż występują zbyt rzadko w tekście. W szczególności może to mieć miejsce jeśli zamiast nazwy własnej występuje zaimek na nią wskazujący. | Wykrywanie nazw własnych pozwala podnieść ich wagę, dzięki czemu trafią do streszczenia. Zastępowanie zaimków również pozwala na lepsze uwzględnianie nazw własnych. |
| Zdania wybrane do streszczenia są pozbawione kontekstu. W szczególności zaimki mogą odnosić się do wyrazów ze zdań, które nie znalazły się w streszczeniu. | Na etapie przetwarzania wstępnego zaimki mogą zostać zastąpione przez wyrazy, do których się odnoszą. W szczególności mogą być zastąpione przez rozpoznane nazwy własne. |
| Jeśli tekst źródłowy składa się z długich zdań, to streszczenie będzie długie lub zbyt mała liczba zdań trafi do streszczenia. | Na etapie przetwarzania wstępnego zdania złożone mogą zostać, w miarę możliwości, podzielone na zdania proste. Dzięki temu zdania wynikowe w streszczeniu mogą być krótsze. Dodatkowo rozbiór zdań może być podstawą do usuwania wybranych wyrazów lub fragmentów zdań. |

### Implementacja

W ramach [Proj16] zaimplementowano zadanie sumaryzacji poprzez ekstrakcję. W ramach projektu stworzono program umożliwiający wygenerowanie streszczeń dokumentów tekstowych w sposób ekstrakcyjny. Wykorzystano kilka algorytmów

umożliwiających wybór najbardziej wartościowych zdań do wynikowego streszczenia. Poniżej przedstawiono zarysy wykorzystanych algorytmów, opisano także funkcjonalność aplikacji oraz wyniki przeprowadzonych testów.

Wejściem programu jest plik tekstowy w formacie XML. Program przetwarza plik wejściowy wyodrębniając z niego poszczególne zdania. W pierwszej fazie zdania są oceniane zgodnie z regułami zdefiniowanymi w pliku konfiguracyjnym (dla wybranej metody sumaryzacji). Każde zdanie otrzymuje wagę. Zadana liczba zdań o najwyższej wadze zostaje wybrana do streszczenia (w kolejności ich występowania w pliku wejściowym). W drugiej fazie przetwarzania może nastąpić eliminacja zdań zbyt do siebie podobnych, aby uniemożliwić powielenia się zdań o podobnej treści w wynikowym streszczeniu.

Możliwe jest zdefiniowanie dla poszczególnych metod sumaryzacji (plik konfiguracyjny) kilku algorytmów, które będą wpływały na wagę zdań. Początkowa waga wszystkich zdań wynosi 1. Każda kolejna waga zdań jest iloczynem wagi poprzedniej oraz wagi wyznaczonej na podstawie algorytmu. Każdy algorytm posiada możliwość zdefiniowania jego wpływu na oceny zdań (atrybut *weight*).

Poniżej przedstawiono sposób działania poszczególnych algorytmów ważenia zdań.

#### Ważenie zdań na podstawie wag słów (TF-IDF)

Algorytm polega na wyliczaniu wag każdego słowa występującego w analizowanym dokumencie. W tym celu wykorzystuje się zbiór tekstów (w formacie XML) umieszczony w zadanym katalogu. Wyliczanie wag słów streszczanego dokumentu oparte jest o algorytm TF-IDF (więcej informacji w rozdziale 2.2.3. Obliczanie wag, str. 23). Ważenie zdań w oparciu o wagi słów przebiega według następującego algorytmu:

$$WAGA\_ZDANIA(I) = \frac{\sum_{J=1}^{N(I)} WAGA\_SLOWA(I,J)}{N(I)}$$

Gdzie:

WAGA_ZDANIA(I) – waga i-tego zdania,

WAGA_SLOWA(I,J) – waga j-tego słowa w i-tym zdaniu,

N(I) – liczba słów w zdaniu i.

### Ważenie zdań na podstawie ich pozycji w dokumencie

W niektórych dokumentach pozycja zdania w dokumencie powinna mieć wpływ na wagę tego zdania. W niektórych wypadkach wybieranie (np. artykuły prasowe) wybieranie początkowych zdań daje wyniki porównywalne do bardziej zaawansowanych algorytmów [Rau, 1994]. Przyjęto założenie, że możliwe jest zwiększanie wag zdań znajdujących się na początku lub końcu dokumentu.

### Ważenie zdań na podstawie tzw. „Cue Phrases"

Możliwe jest zdefiniowanie (w oddzielnym pliku w formacie XML) tzw. *cue phrases*, czyli fraz zwiększających wagę zdań. Okazuje się to przydatne, gdy dla jakiegoś dokumentu znany jest jego tytuł, frazy oraz słowa kluczowe opisujące jego treść lub frazy sugerujące, że zdanie zawiera istotną treść. Występowanie takich fraz w treści zdań może sugerować większe znacznie takiego zdania i powinno mieć większą szansę na znalezienie się w wygenerowanym streszczeniu.

### Ważenie zdań na podstawie długości zdań

Możliwe jest zdefiniowanie zakresu optymalnych długości zdań. Zdania z tego zakresu (np. od 3 do 13 słów) będą wyżej oceniane niż zdania spoza niego. Umożliwia to wyeliminowanie ze streszczenia zbyt długich lub zbyt krótkich zdań. Nowa waga zdania jest iloczynem wagi poprzedniej i wagi zdefiniowanej w pliku konfiguracyjnym.

### Ważenie słów z użyciem tezaurusa

Program wybiera pewną ilość najwyżej ocenionych słów jako słowa określające treść streszczanego dokumentu. Zdania posiadające te słowa lub ich synonimy będą posiadać większą wagę. Synonimy wczytywane są z pliku synonimów dołączonego do aplikacji. Nowa waga zdania jest iloczynem wagi poprzedniej i wagi zdefiniowanej w pliku konfiguracyjnym. W ramach projektu wykorzystano tezaurus stworzony na witrynie http://synonimy.sourceforge.net. W obecnej chwili zawiera on około 22 tysięcy wyrazów.

#### Korzystanie z lematyzatora

Aplikacja umożliwia zdefiniowanie w pliku konfiguracyjnym, czy powinny być używane lematy czy formy odmienione słów przy wyliczaniu wag za pomocą algorytmu TF-IDF.

#### Usuwanie części mowy oraz wyrazów ze streszczenia

Ze zdań, które zostały wybrane do streszczenia możliwe jest usunięcie pewnych zadanych części mowy oraz wyrazów. Elementy, które należy usunąć definiowane są w pliku konfiguracyjnym.

### 4.3.1. Opis testów

Do testów użyto plików testowych przetworzonych na różnych etapach płytkiej analizy tekstu. Testy wykazały , że istotnym z punktu widzenia jak najlepszej i logicznej sumaryzacji jest właściwy dobór opisanych wcześniej parametrów, specyficzny dla różnych typów podsumowywanych dokumentów. W tym celu przeprowadzono eksperymentalny dobór parametrów dla trzech typów dokumentów tekstowych:

- Tekst techniczny
- Powieść
- Artykuł prasowy

Dla każdego z typów dobrano pliki konfiguracyjne, aby jakość sumaryzacji była jak najlepsza.

Dodatkowo przetestowano wpływ wybranych algorytmów doboru wag oraz wstępnego przetwarzania tekstów na jakość generowanych streszczeń. Test przeprowadzono na 4 wybranych losowo dokumentach tekstowych.

Zdefiniowano 3 metody sumaryzacji:

- TEST0
    - Ważenie zdań z wykorzystaniem algorytmu TF-IDF
- TEST1
    - Ważenie zdań z wykorzystaniem algorytmu TF-IDF
    - Karanie zdań posiadających słowa zdefiniowane w pliku konfiguracyjnym

  - Promowanie zdań o odpowiednich długościach
  - Promowanie zdań znajdujących się na początku lub na końcu tekstu
- TEST2
  - Ważenie zdań z wykorzystaniem algorytmu TF-IDF
  - Karanie zdań posiadających słowa zdefiniowane w pliku konfiguracyjnym
  - Promowanie zdań o odpowiednich długościach
  - Promowanie zdań znajdujących się na początku lub na końcu tekstu
  - Korzystanie ze stemming'u podczas wyliczania miar za pomocą algorytmu TF-IDF
  - Dodatkowe promowanie zdań zawierających najwyżej ocenione słowa lub ich synonimy (wykorzystanie tezaurusa języka polskiego)

Na potrzeby testów wyróżniono 3 poziomy płytkiej analizy tekstu:

- P1 – tokenizacja i podział na zdania
- P2 – j.w. oraz analiza morfologiczna i usuwanie niejednoznaczności
- P3 – j.w. oraz rozpoznawanie nazw własnych, zastępowanie zaimków, rozkład zdań złożonych na zdania proste

Wygenerowano streszczenia zawierające 5 zdań. Wyniki znajdują się w załączniku C.

Podstawowym algorytmem wykorzystywanym przez każdą metodę sumaryzacji jest algorytm TF-IDF. Dzięki jego zastosowaniu do streszczenia wybierane są zdania posiadające słowa niosące jakąś „istotną" treść w dokumencie. Jakość działania tego algorytmu zależy głównie od ilości pomocniczych dokumentów, na podstawie których wyliczane są miary „ważności" słów w analizowanych tekstach. W trakcie testów wykorzystano ponad 250 takich tekstów (zbiór tekstów *ONET*). Jednak ich wadą było to, że były one dość krótkie, przez co miary wyliczane na podstawie algorytmu TF-IDF były często niezadowalające. Przykładem dziesięciu najbardziej istotnych słów znajdujących się w tekście Lalki Bolesława Prusa są: „Wokulski", „go", „radca", „szkoły", „roku", „ajent", „s.", „no", „tej", „wariat". Widać, że takie słówka jak „tej"

oraz „go” w praktyce nie powinny być oceniane wysoko, pozostałe słowa zostały jednak znalezione poprawnie.

Jakość działania algorytmu TF-IDF wzrasta znacznie podczas zastosowania stemming'u (po przetworzeniu tekstów posiadających atrybut „lemat”).

Metoda sumaryzacji oparta tylko na wykorzystaniu algorytmu TF-IDF okazała się jednak mało dokładna. Istniało duże ryzyko, że do streszczenia dostaną się zdania zawierające bardzo dużo wyrazów lub bardzo mało. W trakcie testów zdarzało się, że w streszczeniu pojawiło się zdanie „Wariat!”, ze względu na to, że słowo wariat było wysoko oceniane.

Bardzo przydatne w praktyce okazało się zastosowanie promowania zdań, których długość znajduje się w zadanym przedziale. Uniknięto w ten sposób pojawianie się w streszczeniu bardzo długich lub za krótkich zdań. Przykładowo w jednym ze streszczeń pojawiło się zdanie: *„Mimo że obiektywnie sytuacja na rynku pracy niewiele się zmieniła, dzisiejsza młodzież nauczyła się wykorzystywać okazje, jest znacznie bardziej dynamiczna i aktywna- komentuje Tomasz Karoń”*. Mimo, że zdanie niesie dość dużo informacji o treści dokumentu jest długie, przez co być może nie powinno się znaleźć w streszczeniu. Dzięki zastosowaniu tej funkcji zdanie zostało odrzucone ze streszczenia.

Bardziej dyskusyjne okazało się wykorzystanie parametru *usePosition*, dzięki któremu możliwe jest promowanie zdań znajdujących się na początku lub na końcu tekstu. W trakcie tekstów dało się zauważyć, że to założenie może być prawdziwe tylko w niektórych tekstach np. artykułach technicznych lub publikacjach naukowych. Jednak w wielu wypadkach promowanie zdań na początku lub na końcu tekstów daje niewłaściwe rezultaty.

Testy objęły również wpływ promowania lub karania zdań w zależności od pojawiania się w nich pewnych fraz tzw. „***cue phrases***”. Promowanie zdań posiadających takie słowa jak: „podsumowując”, „na zakończenie”, „jest o”, „dotyczy” itp. dawało dobre rezultaty. Okazało się również, że przy streszczeniu takich tekstów jak „Lalka” uzasadnione jest usuwanie ze streszczeń dialogów. Zostało to osiągnięte przy użyciu metody karania zdań za posiadanie pewnych fraz. Okazało się również, że dobre rezultaty daje usuwanie zdań, które są pytaniami. Chociaż w przypadku niektórych

tekstów występowanie pytań w streszczeniach może być uzasadnione, gdy treść tekstu ma formę: krótkie pytanie – długa odpowiedź (np. „Dokąd zmierza chińska nauka").

Wykorzystanie lematyzacji podczas generacji streszczeń również okazało się bardzo dobrym pomysłem. Przede wszystkim wpływa ona pozytywnie na wyliczanie wag za pomocą algorytmu TF-IDF. Lematyzacja ponadto umożliwia większą elastyczność podczas definiowania fraz lub słów w plikach wykorzystywanych w programie. Np. w pliku *cue_phrases.xml* zamiast takich słów jak „podsumowując", „podsumowaniu" itp. możemy wpisać tylko słowo „podsumowanie". Zakładając poprawne działanie lematyzacji poszczególne słowa w tych plikach można pisać tylko w formie podstawowej. Stemming jest również konieczny przy korzystaniu z opisanego niżej promowania zdań z użyciem tezaurusa.

W programie wykorzystano słownik wyrazów bliskoznacznych pobrany z witryny: http://synonimy.sourceforge.net. Dzięki niemu możliwe było poszukiwanie słów bliskoznacznych. Zdania były dodatkowo promowane, jeżeli posiadały słowo z grupy najwyżej ocenionych słów lub jego synonim. Pomysł wykorzystania tego algorytmu zrodził się po przeprowadzeniu pewnej ilości testów. Ważenie zdań na podstawie miar wyznaczonych przez algorytm TF-IDF okazało się trochę niewystarczające – głównie za sprawą niewielkich różnic w wagach pomiędzy zdaniami. Ponadto zwykle to kilka lub kilkanaście słów wystarcza do tego, aby określić temat analizowanego dokumentu.

W programie zaimplementowano również algorytm usuwania ze streszczenia wybranych części mowy oraz konkretnych słów. Jednak wstępne testy pokazały, że nie jest on przydatny. Przede wszystkim nie ma takiej części mowy, którą zawsze należy usuwać ze zdań znajdujących się w streszczeniu. Podobnie jest z konkretnymi wyrazami.

Zaimplementowano również usuwanie ze streszczenia zdań zbyt do siebie podobnych (patrz algorytm MMR).

Trudnym problemem podczas testów jest problem oceny jakości sumaryzacji. W wielu przypadkach bardzo trudno jest stanie ocenić, które z wygenerowanych streszczeń jest najlepsze. Jednak w przeważającej większości przypadków zastosowanie wszystkich zaproponowanych algorytmów dawało najlepsze rezultaty. Warto wspomnieć również o tym, że w małych tekstach zastosowanie stemming'u oraz wykorzystanie tezaurusa

często nie wpływa na zawartość wygenerowanego streszczenia. Największe różnice pojawiają się w streszczeniach wygenerowanych na podstawie pierwszej i drugiej metody. Streszczenie, które powstało na podstawie pierwszej metody zwykle było oceniane jako słabe, głównie z powodu umieszczania w streszczeniu zbyt długich lub zbyt krótkich zdań. Z kolei streszczenia, które powstały przy użyciu metody drugiej i trzeciej zwykle były do siebie podobne. Częściej też lepiej oceniane były streszczenia otrzymane za pomocą trzeciej metody. Jednak były wyjątki! Np. streszczenie wygenerowane za pomocą pierwszej metody:

*„Melanie dochowała się trzech zdrowych córek. Cierpiąca na przedwczesną niewydolność jajników Stephanie przeszła menopauzę już w 14. Pomóc jej próbowała również Melanie. Louis kierowany przez dr Shermana Silbera pobrał jeden z jajników Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować."*

wydaje się dużo lepsze niż streszczenia wygenerowane za pomocą metody 3:

*Siostra pomaga siostrze. Melanie dochowała się trzech zdrowych córek. Jej siostra nie miała tego szczęścia. Pomóc jej próbowała również Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować.*

### 4.3.2. Przykładowe streszczenia

#### Tekst techniczny

Jako reprezentacyjny przykład wybrano plik o nazwie „tech12.xml", który poddano tokenizacji, podziałowi na zdania i analizie morfologicznej. Oprócz ważenia poszczególnych zdań algorytmem TF-IDF zastosowano również bardzo istotne, jak się okazuje, promowanie zdań znajdujących się na początku i na końcu tekstu. Ponieważ tekst nie był dostępny w postaci zdań prostych, mimo zastosowania odpowiedniej kary, do podsumowania dostały się również zdania bardzo długie i złożone. Użyto również promowania zdań zawierających słowa opisujące treść dokumentu wraz z tezaurusem w celu lepszej oceny zdań, co w sposób dość istotny podnosiło jakość sumaryzacji – w szczególności widoczne było to po zróżnicowaniu ocen poszczególnych zdań tekstu.

Podsumowanie wcześniej wymienionego tekstu wygląda następująco:

*Wdrożenie CRM jest przedsięwzięciem ogromnym, bowiem dotyka niemal wszystkich obszarów działalności przedsiębiorstwa. Wystarczy, że zbudujemy je nieco lepiej od naszej konkurencji.*

*Można również zacząć budowę CRM od analizy procesowej, modelowania i przebudowy procesów( na przykład poprzez Business Process Reengineering), oraz– w miarę potrzeb– przekształcić strukturę organizacyjną firmy, której zastany kształt może skutecznie uniemożliwić realizację CRM.*

Widać z powyższego, że główny temat tekstu został uchwycony (głównie dzięki wybraniu zdania z początku i końca tekstu). Natomiast poprawność merytoryczna streszczenia pozostawia jeszcze trochę do życzenia.

### Artykuły prasowe

W tym przypadku konfiguracja sumaryzatora jest bardzo podobna. Bardzo istotne są zdania będące na początku oraz na końcu artykułu (zawierają w sobie najwięcej ogólnych treści). Często nie sposób ominąć niepotrzebnych nagłówków dołączanych do artykułu w pierwszych jego zdaniach – są one nierozróżnialne od pozostałej części tekstu. Ponadto zmieniono długość zdań, która jest optymalna dla streszczenia – w tym przypadku artykuły prasowe mają zdania krótsze niż inne teksty. Oczywiście zastosowano również algorytm TF-IDF. Poniżej przedstawiono streszczenia dwóch wybranych artykułów.

Artykuł 1: „news12.xml”

*Pewna Polka mieszkająca we Włoszech została skazana na cztery miesiące więzienia za udawanie ducha w alpejskim XV- wiecznym zamku Castel Codrano w pobliżu granicy z Austrią i Szwajcarią. Rzekomy duch został złapany na gorącym uczynku. Kobietę skazano za nękanie dyrektorki i straty, spowodowane w zamku.*

Artykuł 2: „453731,12,druk.html.xml”

*Izrael: Kary za odmowę służby na terytoriach palestyńskich IAR, dk/ 2002-01-31 15:47:00 Dowództwo armii izraelskiej ukarało dyscyplinarnie oficerów, którzy w otwartym liście odmówili służby wojskowej na terytoriach palestyńskich. Jak pisze izraelski dziennik" Ha'arec", kilku oficerów- sygnatariuszy listu, już zostało zdegradowanych. Ponadto, major Rami Kaplan przestał być zastępcą dowódcy formacji pancernej. Generał dodał, że tendencja ta stale nasila się w ostatnich latach.*

Z powyższych przykładów widać, że wygenerowane streszczenia, podobnie jak poprzednio zachowują sens i nawet bez znajomości pełnego tekstu, można wywnioskować o jego temacie. Pojawiają się wspomniane problemy z niepotrzebnymi nagłówkami, natomiast spójność językowa jest lepsza niż w przypadku tekstów

technicznych. Warto zauważyć, że artykuły prasowe statystycznie rzecz biorąc były najłatwiejszym typem tekstów do sumaryzacji. Najprawdopodobniej wynika to ze znacznej dostępności przykładów (i ich użycia w algorytmie TF-IDF), ale również z tego, że zdania są dosyć proste i często krótkie, przez co nie trzeba ich rozkładać na zdania proste.

### Powieści

Najtrudniejszym z punktu widzenia rozmiaru (i czasu poświęconego na dobór parametrów) przykładem tekstu okazała się powieść. Problemem były w tym przypadku przede wszystkim dialogi, które bezpośrednio (bez jakichkolwiek zmian) do streszczenia wchodzić nie powinny. W tym przypadku radą na to okazało się zastosowanie opisanych wcześniej fraz, które są promowane. Wykorzystano ten mechanizm nie do promowania, a do karania zdań, które wykazują cechy dialogów. Wyłączono całkowicie funkcja promowania zdań znajdujących się na początku i na końcu tekstu, natomiast wydłużono granice pomiędzy, którymi zdania miały podwyższaną ocenę (od 10 do 20 słów). Użyto również tezaurus, który poprawiał nieznacznie uzyskiwane wyniki.

Dla pliku „lalka1.xml” przetworzonego przez 5 zadań poprzedzających wygenerowane zostało następujące streszczenie:

*W renomowanej jadłodajni, gdzie na wieczorną przekąskę zbierali się właściciele składów bielizny i składów win, fabrykanci powozów i kapeluszy, poważni ojcowie rodzin utrzymujący się z własnych funduszów i posiadacze kamienic bez zajęcia, równie dużo mówiono o uzbrojeniach Anglii, jak o firmie J. Mincel i S. Wokulski. Inni prezydent twierdzili, że Wokulski jest zdecydowanym wariatem, jeżeli nie czymś gorszym... Józiu, przynieś no jeszcze piwa. Dla osób posilających się w tej co radca jadłodajni, dla jej właściciela, subiektów i chłopców przyczyny klęsk mających paść na S. Wokulskiego i jego sklep galanteryjny były tak jasne jak gazowe płomyki oświetlające zakład. Ale żeby Wokulski zdradzał skłonności do wariacji, tegom nie spostrzegł. Radca przyjął cygaro bez szczególnych oznak wdzięczności. I jak dziś Józio, tak on wówczas podawał mi piwo, zrazy nelsońskie... Ten jednak, wariat, rzucił wszystko i pojechał robić interesa na wojnie. Mimo posępne wróżby ludzi trzeźwo patrzących na rzeczy sklep galanteryjny pod firmą J. Mincel i S. Wokulski nie tylko nie upadał, ale nawet robił dobre interesa. Ani jednak ciekawość ogółu, ani fizyczne i duchowe zalety trzech subiektów, ani nawet*

*ustalona reputacja sklepu może nie uchroniłaby go od upadku, gdyby nie zawiadował nim czterdziestoletni pracownik firmy, przyjaciel i zastępca Wokulskiego, pan Ignacy Rzecki.*

Z powyższego streszczenia widać, że bardzo istotnym słowem dla streszczenia stało się nazwisko głównego bohatera (wysoko oceniane algorytmem TF-IDF). Nie można się było również ustrzec zdań z dialogów, które nie pasują kompletnie do reszty streszczenia. Na szczęście zdanie takie występuje w powyższym przykładzie tylko jedno. Wygenerowane streszczenie jest w pewnym stopniu informatywne, natomiast zdania są ze sobą luźno powiązane tematycznie, co jest niewątpliwie wadą. W tym wypadku zastosowanie sumaryzacji ekstrakcyjnej wydaje się niewystarczające.

### 4.3.3. Podsumowanie testów sumaryzacji

Uzyskane wyniki nie są jednoznaczne. W niektórych przypadkach uzyskane streszczenia są bardzo dobre, jednak w niektórych ich jakość jest raczej kiepska. Z drugiej strony nie do końca można określić, jaka metoda sumaryzacji jest najlepsza. Pewne jest jedynie to, że wykorzystanie algorytmu TF-IDF jest pomysłem bardzo dobrym. Stosowanie kolejnych algorytmów już nie zawsze ma uzasadnienie. W dużym stopniu zależy to od charakteru streszczanego dokumentu, rodzaju występujących zdań itp. Dobranie parametrów i metod streszczania do zbioru dokumentów jest sprawą kluczową.

Najbardziej istotnymi etapami płytkiej analizy tekstu, które mogłyby wpływać na jakość generowanych streszczeń okazały się lematyzacja oraz rozkład zdań na zdania proste. W przypadku, gdy metoda wykorzystywała lematyzację oraz w plikach wejściowych istniał dla każdego słowa atrybut „lemat”, wyliczanie wag za pomocą algorytmu TF-IDF było dużo bardziej precyzyjne. W trakcie testów okazało się jednak, że wpływ lematyzacji na proces sumaryzacji nie jest duży. Tak jak wspomniano wcześniej wynika to zarówno z niewielkich rozmiarów streszczanych tekstów jak również z niedoskonałości tekstów wykorzystywanych podczas wyliczania miar za pomocą algorytmu TF-IDF. Znacznie większy okazał się wpływ rozkładu zdania na zdania proste. Dzięki temu zabiegowi dużo mniej zdań było odrzucanych z powodu ich zbyt dużej długości. Również jakoś streszczenia można ocenić jako lepszą.

Jednak wyniki generacji streszczeń przedstawione z załączniku C pokazują, że bardzo trudno jest jednoznacznie określić wpływ płytkiej analizy tekstu na poprawę jakości

streszczeń. Wydaje się, że większy wpływ na dobór odpowiednich parametrów algorytmu sumaryzacji.

Wygenerowane w trakcie testów streszczenia mają wartość informatywną, jednak ich jakość i spójność w wielu przypadkach jest niewystarczająca. W dużym stopniu wynika to z zastosowanego podejścia, tzn. sumaryzacji ekstrakcyjnej. Dla niektórych tekstów wybieranie zdań z tekstu źródłowego nie pozwala na uzyskanie zadowalającego streszczenia nawet w przypadku, gdy zdania są wybierane przez eksperta. Wydaje się więc, że aby uzyskać lepszą jakość streszczeń konieczne jest zastosowanie podejścia generacyjnego bazującego na pełniejszej reprezentacji tekstu oraz korzystającym z dodatkowej bazy wiedzy, np. w postaci ontologii. Dobrym kierunkiem badań jest więc wykorzystanie wyników płytkiej analizy tekstu do budowy złożonych reprezentacji tekstu oraz ontologii.

# 5. Podsumowanie

W niniejszej rozprawie przeanalizowano wybrane zagadnienia eksploracji tekstów w kontekście ich zastosowania do przetwarzania dokumentów w języku polskim. W ramach wprowadzenia do dziedziny przedstawiono i omówiono najczęściej spotykane zadania eksploracji tekstów. Zaprezentowano stosowane reprezentacje tekstów i metody ich przetwarzania, a także najpopularniejsze algorytmy klasyfikacji i grupowania dokumentów. Poruszono również temat sumaryzacji dokumentów.

Szerzej zbadano metody płytkiej analizy tekstu, która łączy w sobie podejścia klasycznej analizy lingwistycznej NLP oraz technik eksploracji tekstów wywodzących się z metod statystycznych. Autor niniejszej rozprawy zbadał wykorzystanie modelu płytkiej analizy dokumentów tekstowych w języku polskim i zaproponował konkretne algorytmy i rozwiązania wydzielonych z niego zadań. Pozwoliło to zaprojektować platformę badawczą i w jej ramach stworzyć zestaw narzędzi. Główna część pracy obejmuje szczegółowe przedstawienie różnych aspektów kolejnych etapów płytkiej analizy tekstu oraz możliwych podejść. W szczególności omówiono zadania:

- tokenizacji,
- analizy morfologicznej,
- usuwania niejednoznaczności,
- rozpoznawania nazw własnych,
- zastępowania zaimków,
- rozkładu zdań prostych na złożone,
- rozbioru zdań.

Przedstawiono również wyniki implementacji i eksperymentów oraz omówiono wady i zalety zastosowanych rozwiązań. Stworzone narzędzia mogą służyć jako przydatna platforma do dalszych badań nad narzędziami automatycznego przetwarzania tekstów w języku polskim. Dzięki otwartej strukturze platformy możliwe jest dalsze rozwijanie każdego z jej elementów osobno. Ma to szczególne znaczenie, gdyż ze względu na szeregowe przetwarzanie błędy na kolejnych etapach przetwarzania mogą się

kumulować, więc ograniczenie liczby błędów na dowolnym etapie będzie prowadzić do zwiększenia poprawności całego przetwarzania.

Aby zbadać możliwości praktycznego wykorzystania efektów płytkiej analizy tekstu przeprowadzono eksperymenty wykorzystania dodatkowych informacji lingwistycznych do poprawy wyników klasyfikacji i grupowania tekstów. Zaproponowany eksperyment obejmował zadanie klasyfikacji i grupowania rzeczywistego zbioru danych. Wykorzystano popularne algorytmy, jednak wzbogacone o wykorzystanie dodatkowych informacji pochodzących z płytkiej analizy tekstu. Testy pokazały, że zastosowane metody pozwalają na uzyskiwanie lepszych wyników. Mimo, że poprawa nie jest duża, to jednak jest na tyle znacząca, że pozwala wnioskować o potencjale stosowania tych metod.

W chwili obecnej praktyczne wykorzystanie tego typu technik w narzędziach komercyjnych jest wciąż dosyć niskie. Należy się jednak spodziewać, że ich wykorzystanie będzie rosło. W szczególności powinno objąć wyszukiwarki internetowe i inne systemy publikacji i wyszukiwania informacji, co pozwoli użytkownikom zadawać pytania w sposób bardziej naturalny i uzyskiwać precyzyjniejsze wyniki.

Dodatkowo poruszono temat automatycznej sumaryzacji tekstów w języku naturalnym. Sam problem sumaryzacji jest bardzo złożony i do tej pory nie zostały zaproponowane uniwersalne metody jego rozwiązania. Stosowane metody opierają się głównie na ekstrakcji fragmentów tekstu źródłowego. W ramach prac badawczych autora przeprowadzono eksperymenty z implementacją systemu sumaryzacyjnego, a przykładowe wyniki zaprezentowana w niniejszej rozprawie.

Potencjalnie wprowadzenie narzędzi analizy lingwistycznej może nieco poprawić jakość sumaryzacji. Sprawdzona możliwości zastosowania narzędzi płytkiej analizy tekstu do poprawy jakości sumaryzacji. Uzyskane wyniki nie są jednoznaczne. Na pojedynczych przykładach elementy płytkiej analizy tekstu eliminują pewne ograniczenia metody ekstrakcyjnej. Jednak test na danych rzeczywistych nie wykazał znaczącej poprawy. Wiąże się to m.in. z trudnością oceny jakości sumaryzacji. Jak się okazuje sumaryzacja jest procesem wymagającym największej ilości informacji lingwistycznych i dlatego jest najbardziej wrażliwa na wszelkie nieprawidłowości w procesie płytkiej analizy tekstu.

Podsumowując płytka analiza tekstu w języku polskim jest możliwa do realizacji i uzyskiwane wyniki są zadowalające. Możliwe jest jej zastosowanie do zwiększenia poprawności zadań eksploracji tekstu, takich jak: klasyfikacja, grupowanie czy sumaryzacja.

Dalsze kierunki badań powinny obejmować:

- zwiększenie poprawności i skuteczności stworzonych metod i narzędzi,
- zastosowanie płytkiej analizy tekstu w zadaniach wyszukiwania i ekstrakcji informacji,
- zastosowanie płytkiej analizy tekstu do rozwiązywania innych praktycznych zadań eksploracji tekstu,
- wsparcie procesu automatycznej budowy słowników, tezaurusów i ontologii z wykorzystaniem opracowanych metod,
- stworzenie metod sumaryzacji dokumentów w oparciu o płytką analizę tekstu oraz informacje z ontologii.

# 6. Załącznik A: Środowisko implementacyjne

W celu przeprowadzenia eksperymentów z zastosowaniem płytkiej analizy tekstów do przetwarzania tekstów w języku polskim zostało zaprojektowane przez autora niniejszej rozprawy środowisko implementacyjne. Wydzielono następujące zadania:

- Zad1: Tokenizacja i wykrywanie końca zdania;
- Zad2: Rozpoznawanie części mowy (analiza morfologiczna);
- Zad3: Usuwanie niejednoznaczności;
- Zad4: Rozpoznawanie zaimków i nazw własnych;
- Zad5: Rozkład zdań złożonych na zdania proste;
- Zad6: Rozbiór zdania.

Zadania 1-4 są częściowo zgodne z podziałem zadań zaproponowanym w [Piskorski, 2001]. Jednak w odróżnieniu od tej pracy wykrywanie końca zdania nie zostało wydzielone jako oddzielne zadanie, lecz włączone do tokenizacji. Zmieniona została również kolejność zadań na bardziej pasującą do zastosowanego podejścia. Wykrywanie końca zdania jest wykonywane na poziomie informacji o interpunkcji i układzie tekstu oraz nie korzysta z informacji o morfologii wyrazów.

Zadania 5 i 6 zostały zaproponowane przez autora niniejszej pracy.

## *6.1. Format zapisu tokenów*

Istotnym aspektem jest uniwersalny i elastyczny format zapisu tokenów. W celu zachowania dużej niezależności od platformy implementacyjnej narzędzi realizujących dalsze etapy przetwarzania STP ważne jest stosowanie łatwo przenośnego formatu. Dlatego wybraną podstawą definiowania formatu zapisu tokenów stał się standard XML. W celu zachowania możliwości obsługi w przyszłości innych języków, a także przetwarzania dokumentów wielojęzycznych została podjęta decyzja o zapisie znaków z standardzie Unicode UTF-8.

Tokeny są numerowane kolejno od 1. Każdy token oprócz ustalonych atrybutów zawiera atrybut *„słowo”* z oryginalnym brzmieniem słowa (forma odmieniona). Dodatkowo dla znaków interpunkcyjnych, znaków specjalnych oraz słów nierozpoznanych są tworzone tokeny, które zawierają jedynie atrybuty: *id*, *słowo*, *rodzajTokena*, np.:

```
<token id=”7” slowo=”:” rodzajTokena=”znak” />
```

Służy to przeprowadzaniu operacji odwrotnej, czyli odtwarzaniu tekstu pierwotnego.

Poniżej znajduje się opis formatów plików na każdym z etapów przetwarzania (z różnych zadań). Zadanie nr 1 tworzy plik XML na podstawie pliku tekstowego. W miarę możliwości kolejne zadania uzupełniają plik XML bez utraty zawartych już w nim informacji poprzez dołożenie kolejnych atrybutów. Wyjątkiem jest np. podział zdań złożonych na proste lub usuwanie niejednoznaczności.

Dodatkowo w pliku XML występuje sekcja <*historia*>, gdzie zapisywane są informacje o tym, jakie zadania przetwarzały już danych plik. Atrybut *czas* zawiera czas przetwarzania w sekundach. Atrybut *data*, zawiera informację o momencie zakończenia przetwarzania.

*UWAGA: Nazwy atrybutów i ich wartości zawierają tylko małe litery i nie zawierają polskich liter.*

### 6.1.1. Zad1: Tokenizacja i wykrywanie końca zdania

W tym zadaniu na podstawie danych wejściowych w postaci pliku .txt zawierającego tekst w języku polskim tworzony jest plik wyjściowy XML zawierający tokeny pogrupowane w zdania. Znaki przystankowe powinny być oddzielnymi tokenami. Tokeny powinny być numerowane globalnie w ramach pliku (nie w ramach zdania).

Tokeny są numerowane od 1. Numeracja musi być spójna, tzn. token występuje nawet jeśli dane słowo nie jest przetwarzane przez zadanie.

Atrybut zdanie.kontekst:

- Zwykly – normalny tekst
- Naglowek – naglowek tekstu
- Lista – element listy, wyliczenia itp.
- Dialog – fragment dialogu
- Cytat – element cytatu

- Tabela – komórka tabeli

Atrybut token.rodzajTokena (w nawiasie przykłady tokenów):

- liczbaNaturalna („1234”, „433 322”)
- liczbaRzeczywista („123,53”, „134.32”, „125 432,76” )
- liczbaProcent (23%)
- liczbaSymbol (234PLN, 987EUR, 100cm, $100, USD255)
- data (10/12/2004, 10-12-2004, 10.12.2004)
- godzina (10:30, 10.30)
- slowoDuze (ABC, CMS)
- slowoMale (projekt, komin)
- slowoPierwszaDuza (Politechnika)
- slowoMieszane (PGNiG)
- slowoApostrof (Jacksons’, Jerry’s)
- slowoMyslnik (-ność, dostatecz-)
- slowoSymbol (K&M, A+B, /text)
- skrot (potencjalnie skrót: “mgr.”, „abc.”, „m.in.”, „cddfs.”, „S.A”)
- znak – znak interpunkcyjny, może również składać się z kilku znaków, np. wielokropek
- email (abc@cdf.pl)
- url (http://www.abc.pl, www.abc.pl, https://abc.pl)
- CR – znak końca linii (do odtwarzania tekstu)
- tab – znak tabulacji

**Przykładowy plik tekstowy:**

```
Ala ma kota. Plan dnia:
8:00 śniadanie
21:00 kolacja
```

**Dokument XML będący wynikiem zadania 1:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<dokument>
	<historia>
		<zadanie nr=”1” nazwa=”tokenizacja” wersja=”Kowalski”
			czas=”45” data=”2004.12.15 16:44”>
	</historia>
	<zdanie id=”1” kontekst=”zwykly”>
		<token id=”1” slowo=”Ala” rodzajTokena=”slowo” />
		<token id=”2” slowo=”ma” rodzajTokena=”slowo” />
		<token id=”3” slowo=”kota” rodzajTokena=”slowo” />
```

```
            <token id=”4” slowo=”.” rodzajTokena=”znak” />
      </zdanie>
      <zdanie id=”2” kontekst=”zwykly”>
            <token id=”5” slowo=”Plan” rodzajTokena=”slowo” />
            <token id=”6” slowo=”dnia” rodzajTokena=”slowo” />
            <token id=”7” slowo=”:” rodzajTokena=”znak” />
            <token id=”8” slowo=”” rodzajTokena=”CR” />
      <zdanie>
      <zdanie id=”3” kontekst=”lista”>
            <token id=”9” slowo=”8:00” rodzajTokena=”godzina” />
            <token id=”10” slowo=”śniadanie” rodzajTokena=”slowo” />
            <token id=”11” slowo=”” rodzajTokena=”CR” />
      <zdanie>
      <zdanie id=”4” kontekst=”lista”>
            <token id=”12” slowo=”21:00” rodzajTokena=”godzina” />
            <token id=”13” slowo=”kolacja” rodzajTokena=”slowo” />
            <token id=”14” slowo=”” rodzajTokena=”CR” />
      <zdanie>
</dokument>
```

### 6.1.2. Zad2: Rozpoznawanie części mowy

W tym zadaniu tokeny są uzupełniane o dodatkowe atrybuty zawierające informacje z analizy morfologicznej zgodnie ze specyfikacją podaną poniżej. Powinno być możliwe wielokrotne przetwarzanie pliku przez zadanie 2, np. przez różne wersje implementacji zadania. Dzięki temu jedna implementacja może uzupełniać informacje, których nie wprowadziła inna implementacja. Z tego powodu artybuty mogą być tylko dodawane lub modyfikowane, ale nie usuwane.

```
::TOKEN::<unique_id>
Lemat: <podstawa słowotwórcza>
Część mowy: <Rzeczownik|Czasownik|Przymiotnik|.... >
:Rzeczownik:
            Przypadek : <Mianownik | Dopełniacz | ...>
            Rodzaj: <Żeński|Męski|Nijaki}Męskoosobowy|Niemęskoosobowy>
            Liczebność: <Pojedyńczy|Mnogi>
      :Czasownik:
            Czas: <Przeszly|Teraźniejszy|Przyszły|Bezokolicznik>
```

```
            Liczba:  <Pojedyncza|Mnoga>
            Osoba:  <Pierwsza|Druga|Trzecia>
            Rodzaj: <Meski|Żeński|Nijaki|Meskoosobowy|Niemęskoosobowy>
            Tryb: <Rozkazujący|Przypuszaczający|Orzekający>
            Strona: <Bierna|Czynna>
            +Lewy: ::Rzeczownik::<unique_id>
      :Przymiotnik:
            Typ: <Jakościowy|Relacyjny>
            Stopień: <Podstawowy|Wyższy|Najwyższy>
            Przypadek: <Mianownik | ...>
            Rodzaj: <Żeński|Męski|Nijaki>
            +Prawy: ::Rzeczownik::<unique_id>
      :Liczebnik:
            Typ: <Ilościowy|Porządkowy>
                  :Ilościowy:
                        PodTyp:
<główny|zbiorowy|ułamkowy|wielokrotny|wieloraki|mnożny>
                  :Porządkowy:
            Typ2: <Określony|Nieokreślony>
      :Zaimek:
            Typ: <Osobowy|Zwrotny|Pytajno-
względny|Przeczący|Nieokreślony|Wskazujący>
            Przypadek: <Mianownik| ...>
            +Liczba: <Pojedyńca|Mnoga>
            +Rodzaj: <Żeński|Męski|Nijaki>
            +Lewy: (kto?)
            +Prawy: (co?)
      :Przysłówek:
            Typ: <pierwotny|wtórny>
            Stopień: <Podstawowy|Wyższy|Najwyższy>
      :Przyimek:
            Typ: <pierwotny|wtórny>
            +Prawy: (czym?)
      :Spójnik:
            Typ: <Współrzędności|Podrzędności>
                  :Współrzędności:
Podtyp: <Łączny|Rozłączny|Wynikowy|Przeciwstawny|Włączny>+
      :Modulant:
            Typ: <Sytuujący|Waloryzujący|Modalny|Afektujący|Wprowadzający>
      :Wykrzyknik:
            Typ: <Pierwotny|Wtórny>
      :Nakaźnik:
```

```
            Typ: <Pierwotny|Wtórny>
```

UWAGA: Nazwy atrybutów i ich wartości zawierają tylko małe litery i nie zawierają polskich liter. Wyjątek stanowią wartości atrybutów *slowo* i *lemat*.

**Dokument XML będący wynikiem zadania 2:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<dokument>
      <historia>
            <zadanie nr=”1” nazwa=”tokenizacja” wersja=”Kowalski”
                  czas=”45” data=”2004.12.15 16:44”>
            <zadanie nr=”2” nazwa=”morfologia” wersja=”Kowalski”
                  czas=”36” data=”2004.12.15 16:48”>
      </historia>
      <zdanie id=”1” kontekst=”zwykly”>
            <token id=”1” slowo=”Ala” rodzajTokena=”slowo” lemat=”Ala”
czescMowy=”rzeczownik” przypadek=”mianownik” rodzaj=”zenski”
liczebnosc=pojedynczy”/>
            <token id=”2” slowo=”ma” rodzajTokena=”slowo” lemat=”mieć”
czescMowy=”czasownik” liczba=”pojedyncza” osoba=”trzecia” rodzaj=”meski”
tryb=”orzekajacy” czas=terazniejszy” strona=”czynna” lewy=”1”/>
            <token id=”2” slowo=”ma” rodzajTokena=”slowo” lemat=”mieć”
czescMowy=”czasownik” liczba=”pojedyncza” osoba=”trzecia” rodzaj=”zenski”
tryb=”orzekajacy” czas=terazniejszy” strona=”czynna” lewy=”1”/>
            <token id=”2” slowo=”ma” rodzajTokena=”slowo” lemat=”mieć”
czescMowy=”czasownik” liczba=”pojedyncza” osoba=”trzecia” rodzaj=”nijaki”
tryb=”orzekajacy” czas=terazniejszy” strona=”czynna” lewy=”1”/>
            <token id=”3” slowo=”kota” rodzajTokena=”slowo” lemat=”kot”
czescMowy=”rzeczownik” przypadek=”dopelniacz” rodzaj=”meski”
liczebnosc=”pojedynczy”/>
            <token id=”3” slowo=”kota” rodzajTokena=”slowo” lemat=”kot”
czescMowy=”rzeczownik” przypadek=”biernik” rodzaj=”meski”
liczebnosc=”pojedynczy”/>
            <token id=”4” slowo=”.” rodzajTokena=”znak” />
      </zdanie>
      <zdanie id=”2” kontekst=”zwykly”>
            <token id=”5” slowo=”Plan” rodzajTokena=”slowo” lemat=”plan”
czescMowy=”rzeczownik” przypadek=”mianownik” rodzaj=”meski”
liczebnosc=”pojedynczy”/>
            <token id=”5” slowo=”Plan” rodzajTokena=”slowo” lemat=”plan”
czescMowy=”rzeczownik” przypadek=”biernik” rodzaj=”meski”
liczebnosc=”pojedynczy”/>
            <token id=”6” slowo=”dnia” rodzajTokena=”slowo” lemat=”dzień”
czescMowy=”rzeczownik” przypadek=”dopelniacz” rodzaj=”meski”
liczebnosc=”pojedynczy”/>
            <token id=”7” slowo=”:” rodzajTokena=”znak” />
```

```
            <token id=”8” slowo=”” rodzajTokena=”CR” />
      <zdanie>
      <zdanie id=”3” kontekst=”lista”>
            <token id=”9” slowo=”8:00” rodzajTokena=”godzina” />
            <token id=”10” slowo=”śniadanie” rodzajTokena=”slowo”
lemat=”śniadanie” czescMowy=”rzeczownik” przypadek=”mianownik” rodzaj=”meski”
liczebnosc=”pojedynczy”/>
            <token id=”10” slowo=”śniadanie” rodzajTokena=”slowo”
lemat=”śniadanie” czescMowy=”rzeczownik” przypadek=”biernik” rodzaj=”meski”
liczebnosc=”pojedynczy”/>
            <token id=”11” slowo=”” rodzajTokena=”CR” />
      <zdanie>
      <zdanie id=”4” kontekst=”lista”>
            <token id=”12” slowo=”21:00” rodzajTokena=”godzina” />
            <token id=”13” slowo=”kolacja” rodzajTokena=”slowo”
lemat=”kolacja” czescMowy=”rzeczownik” przypadek=”mianownik” rodzaj=”zenski”
liczebnosc=”pojedynczy”/>
            <token id=”14” slowo=”” rodzajTokena=”CR” />
      <zdanie>
</dokument>
```

*UWAGA**:** Token 2 wskazuje na token 1 poprzez atrybut „lewy”. Implementacja nie musi umieszczać takich informacji, gdyż mogą one być uzupełniane na dalszych etapach przetwarzania.*

*UWAGA 2**:** Proszę zwrócić uwagę na występowanie wielu wersji tokenów 2,3,5,10 ze względu na niejednoznaczność.*

### 6.1.3. Zad3: Usuwanie niejednoznaczności

Zadanie polega na usuwaniu niejednoznaczności, które pojawiają się na poziomie analizy morfologicznej poprzez wybieranie odpowiedniego wariantu danego tokenu. Na wejściu podawany jest wynik z zad2. Format jest taki jak w przypadku zadania 2, ale występują już tylko pojedyncze poprawne formy tokenów, które w zadaniu 2 zostały uznane za wieloznaczne.

**Dokument XML będący wynikiem zadania 3:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<dokument>
      <historia>
```

```
            <zadanie nr="1" nazwa="tokenizacja" wersja="Kowalski"
                  czas="45" data="2004.12.15 16:44">

            <zadanie nr="2" nazwa="morfologia" wersja="Kowalski"
                  czas="36" data="2004.12.15 16:48">

            <zadanie nr="3" nazwa="niejednoznaczość" wersja="Kowalski"
                  czas="44" data="2004.12.15 16:50">

      </historia>

      <zdanie id="1" kontekst="zwykly">

            <token id="1" slowo="Ala" rodzajTokena="slowo" lemat="Ala"
czescMowy="rzeczownik" przypadek="mianownik" rodzaj="zenski"
liczebnosc=pojedynczy"/>

            <token id="2" slowo="ma" rodzajTokena="slowo" lemat="mieć"
czescMowy="czasownik" liczba="pojedyncza" osoba="trzecia" rodzaj="zenski"
tryb="orzekajacy" czas=terazniejszy" strona="czynna" lewy="1"/>

            <token id="3" slowo="kota" rodzajTokena="slowo" lemat="kot"
czescMowy="rzeczownik" przypadek="dopelniacz" rodzaj="meski"
liczebnosc="pojedynczy"/>

            <token id="4" slowo="." rodzajTokena="znak" />

      </zdanie>

      <zdanie id="2" kontekst="zwykly">

            <token id="5" slowo="Plan" rodzajTokena="slowo" lemat="plan"
czescMowy="rzeczownik" przypadek="mianownik" rodzaj="meski"
liczebnosc="pojedynczy"/>

            <token id="6" slowo="dnia" rodzajTokena="slowo" lemat="dzień"
czescMowy="rzeczownik" przypadek="dopelniacz" rodzaj="meski"
liczebnosc="pojedynczy"/>

            <token id="7" slowo=":" rodzajTokena="znak" />

            <token id="8" slowo="" rodzajTokena="CR" />

      <zdanie>

      <zdanie id="3" kontekst="lista">

            <token id="9" slowo="8:00" rodzajTokena="godzina" />

            <token id="10" slowo="śniadanie" rodzajTokena="slowo"
lemat="śniadanie" czescMowy="rzeczownik" przypadek="mianownik" rodzaj="meski"
liczebnosc="pojedynczy"/>

            <token id="11" slowo="" rodzajTokena="CR" />

      <zdanie>

      <zdanie id="4" kontekst="lista">

            <token id="12" slowo="21:00" rodzajTokena="godzina" />

            <token id="13" slowo="kolacja" rodzajTokena="slowo"
lemat="kolacja" czescMowy="rzeczownik" przypadek="mianownik" rodzaj="zenski"
liczebnosc="pojedynczy"/>

            <token id="14" slowo="" rodzajTokena="CR" />

      <zdanie>

</dokument>
```

### 6.1.4. Zad4: Rozpoznawanie zaimków i nazw własnych

Format plików jest analogiczny do zadań 1-3. Dodatkowo umieszczane są informacje o rozpoznanych zaimkach i nazwach własnych.

Jeśli dany token został rozpoznany jako nazwa własna, to dodawany jest atrybut *token.nazwatyp* i ewentualnie *token.nazwapodtyp*, które zawierają jako wartość typ i podtyp nazwy własnej:

- osoba
  - imie
  - nazwisko
  - tytul
  - ...
- miejsce
  - panstwo
  - miasto
  - dzielnica
  - ...
- czas
  - godzina
  - data
  - wiek (stulecie)
  - ...
- organizacja
  - firma
  - instytycja
  - ...
- waluta
- procent
- adres
  - ulica (z numerem), kodPocztowy, email, url, tel, fax, kontoBankowe
- ... (inne zaproponowane przez zespół)

**UWAGA**: Należy zwrócić uwagę, że wiele tokenów może wspólnie tworzyć jedną nazwę własną i wówczas wszystkie tokeny powinny być oznaczone tym samym typem, ewentualnie elementy tej samej nazwy własnej mogą mieć różne podtypy, np.: „ABC Sp. z o.o.”, „Pan mgr. inż. Jan Kowalski”, „Andersa 13, 00-159 Warszawa”.

W przypadku rozpoznawania zaimków znajdywany jest token, na który wskazuje dany zaimek i ustawienie odpowiedniego odnośnika w postaci atrybutu *lewy* lub *prawy* (w zależności od pytania, ma które odpowiada). Jeśli zaimek wskazuje na nazwę własną

składającą się z wielu tokenów, to powinien on wskazywać na pierwszy token tej nazwy własnej.

**Przykład:**

```
...
<token id=”1” slowo=”Ala” rodzajTokena=”slowo” lemat=”Ala”
czescMowy=”rzeczownik” przypadek=”mianownik” rodzaj=”zenski”
liczebnosc=pojedynczy” nazwatyp=”osoba” nazwapodtyp=”imie”/>
...
<token id=”7” slowo=”Ona” rodzajTokena=”slowo” lemat=”on” czescMowy=”zaimek”
przypadek=”mianownik” rodzaj=”zenski” liczebnosc=pojedynczy” lewy=”1”/>
...
```

### 6.1.5. Zad5: Rozkład zdań złożonych na zdania proste

To zadanie modyfikuje zarówno układ zdań, jak i tokenów. Z tego powodu może zmienić się numeracja tokenów i zdań. Konieczne jest więc przenumerowanie wszystkich odnośników do tokenów.

W pliku wynikowym do znaczników <*zdanie*> dodawany jest nowy atrybut *typZdania*. Jego wartość określa rozpoznany rodzaj zdania zarówno w przypadku zdań zmodyfikowanych (rozłożonych na proste), jak i niezmodyfikowanych.

Możliwe wartości artybutu zdanie.typZdania:

- proste
- laczne
- rozlaczne
- wynikowe
- przeciwstawne
- podmiotowe
- orzecznikowe
- dopelnieniowe
- przydawkowe
- okolicznikowe

### 6.1.6. Zad6: Rozbiór zdania

Format wynikowy ma strukturę hierarchiczną zgodną z hierarchicznym drzewem rozbioru zdania. W tym celu używany jest znacznik <*grupa*> z atrybutem rodzajGrupy. Atrybut ten może przyjmować następujące wartości:

- grupaPodmiotu
- grupaOrzeczenia
- .....
- podmiot
- orzeczenie
- przydawka
- .....

Znacznik <*grupa*> może zawierać również atrybut *typ*, który zawiera dodatkową informację, np. typ orzeczenia: imienne, osobowe itp.

Każdy znacznik <*zdanie*> może zawierać dowolną liczbę znaczników <*grupa*>, która zagnieżdżając się tworzą strukturę drzewiastą. W wewnątrz znacznika <*grupa*> na dowolnym poziomie są zagnieżdżane znaczniki <*token*>. W miarę możliwości zachowywany jest początkowy porządek tokenów, chociaż nie we wszystkich sytuacjach jest to możliwe.

# 7. Załącznik B: Krótki opis części zdania w j. polskim

W gramatyce języka polskiego wyróżniamy następujące części zdania:

- Podmiot
- Orzeczenie
- Przydawka
- Dopełnienie
- Okolicznik

## *7.1. Podmiot*

Podmiot może być wyrażony każdą częścią mowy *(Uczeń czyta. Najlepszy odpowiada. Pierwszy wygrywa. On zostaje. Siedzieć na lekcji jest nudno. Wysoko to przysłówek odprzymiotnikowy.).*

**Podmiot gramatyczny** odpowiada na pytania:

- Mianownika (kto? co?) *(Uczeń czyta.).*

**Podmiot logiczny** odpowiada na pytania:

- Dopełniacza (kogo? czego?) *(Ucznia nie ma.)*
- Celownika (komu? czemu?) *(Nie chce mi się czekać.)*

**Podmiot szeregowy** tworzą co najmniej dwa wyrazy *(Nauczyciel i uczniowie rozpoczęli lekcje.).*

**Podmiot domyślny** występuje wówczas, gdy tworzą go zaimki 1. i 2. osoby obu liczb *(Ja idę, a ty biegniesz.)* lub gdy jest on częścią orzeczenia *(Zrobimy to!)*

**Zdania bezpodmiotowe**, to takie zdania, które nie zawierają formalnego podmiotu (Świta. Dnieje. Ściemniło się. Czy podać herbatę?).

## *7.2. Orzeczenie*

Orzeczenie określa nazwę czynności lub stanu.

**Orzeczenie czasownikowe** wyrażone jest osobową formą czasownika *(Uczniowie czytają. Uczniowie uczą się. On by napisał.).*

**Orzeczenie imienne** to połączenie rzeczowników i przymiotników (rzadziej imiesłowów, zaimków, liczebników) – czyli orzeczników – z czasownikami zwanymi

łącznikami (być, bywać, stać się, stawać się, zostać, zrobić się) *(On jest uczniem. Ona bywa mądra. Oni są myślący.*).

**Orzeczenie modalne** to połączenie czasowników typu „chcieć", „móc", „musieć", „raczyć", „zamierzać" z bezokolicznikiem *(On chce wyjść. Ona musi czytać. My możemy pomóc.)*

Orzeczenie może być wyrażone **nieosobową formą czasownika** – nie wskazuje wówczas wykonawcy *(Wysprzątano sale. Oddano prace.).*

## *7.3. Przydawka*

Określa wyłącznie rzeczownik (zaimek rzeczowny).

Przydawka przymiotna wyrażona jest:

- przymiotnikiem (jaka?) *(dobra książka),*
- imiesłowem przymiotnikowym (jaka?) *(interesująca książka),*
- liczebnikiem (która?) *(pierwsza książka),*
- liczebnikiem (ile?) *(dwie książki),*
- zaimkiem (czyja?) *(moja książka).*

**Przydawka dopełniaczowa** to rzeczownik w dopełniaczu (kogo? czego?) *(książka koleżanki).*

**Przydawka przyimkowa** ma postać wyrażenia przyimkowego (z czego?) *(książka z biblioteki).*

**Przydawka rzeczowna** to rzeczownik w tym samym przypadku, co wyraz określany (co?) *(książka lektura*).

**Przydawka okolicznościowa** przypomina okolicznik, określa rzeczownik (jak?) *(czytanie na głos).*

## *7.4. Dopełnienie*

Dopełnienie uzupełnia treści orzeczenia. Odpowiada na pytania przypadków zależnych.

**Dopełnienie bliższe** może zostać w zdaniu podmiotem *(Marek czyta* (kogo? co?) *książkę.* → (kto? co?) *Książka jest czytana przez Marka).*

**Dopełnienie dalsze** nie może zostać podmiotem *(Marek patrzy na* (kogo? co?) *tablicę).*

## *7.5. Okolicznik*

Okolicznik określa wyłącznie orzeczenie.

**Okolicznik miejsca** odpowiada na pytania: gdzie? skąd? dokąd? którędy? (Siedzi *tutaj*. Wyszedł *ze szkoły*. Poszedł *do biblioteki*. Idzie *przez* *boisko*.).

**Okolicznik czasu** odpowiada na pytania kiedy? jak długo? jak często? *(Czytam teraz. Uczę się rok. Czytam codziennie.).*

Okolicznik sposobu odpowiada na pytania jak? jaki sposób? *(Czyta płynie. Zachowuje się jak dorosły.).*

**Okolicznik przyczyny** odpowiada na pytania dlaczego? z jakiego powodu? *(Nie przeczytał przez nieuwagę. Opuścił z powodu choroby.).*

**Okolicznik celu** odpowiada na pytania po co? w jakim celu? (Pisze *ku pokrzepieniu serc*. Przebrał się *dla niepoznaki*.).

**Okolicznik warunku** odpowiada na pytanie pod jakim warunkiem? *(Uczył się przy głośnej muzyce.).*

**Okolicznik stopnia i miary** odpowiada na pytania jak bardzo? ile? *(Tańczył do upadłego. Ostrzegała wielokroć.).*

**Okolicznik przyzwolenia** odpowiada na pytanie mimo co? *(Odpisywał mimo zakazu*.).

# 8. Załącznik C: Wyniki sumaryzacji dokumentów

W tym załączniku znajdują się wyniki testów opisanych w 4.3.1. Opis testów (str. 122)

**Bankowa superfuzja - UniCredito przejmuje HVB**

| | P1 | P2 | P3 |
|---|---|---|---|
| **TEST0** | Połączenie ma przynieść obu bankom od 2008 r. miliard euro oszczędności rocznie. Włosi oddali HVB nadzór nad przyszłą fuzją swoich polskich banków- Pekao i BPH. UniCredito zaproponuje też odkupienie akcji BA-CA i BPH od mniejszościowych akcjonariuszy. Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. | Połączenie ma przynieść obu bankom od 2008 r. miliard euro oszczędności rocznie. Włosi oddali HVB nadzór nad przyszłą fuzją swoich polskich banków- Pekao i BPH. UniCredito przejmie od głównych akcjonariuszy HVB również akcje austriackiego banku BA-CA oraz odkupi kontrolny pakiet krakowskiego BPH, najważniejszego banku należącego do niemieckiej grupy w Europie Środkowej. Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. | Połączenie ma przynieść obu bankom od 2008 r. miliard euro oszczędności rocznie. Włosi oddali HVB nadzór nad przyszłą fuzją swoich polskich banków- Pekao i BPH. UniCredito przejmie od głównych akcjonariuszy HVB również akcje austriackiego banku BA-CA oraz odkupi kontrolny pakiet krakowskiego BPH, najważniejszego banku należącego do niemieckiej grupy w Europie Środkowej. Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. |
| **TEST1** | Pracę może stracić 10 tys. Co ważniejsze, stanie się niekwestionowanym liderem w Europie Środkowej. Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. Odgrażają się też polskie banki. | Pracę może stracić 10 tys. osób. 500 zł, na giełdzie papiery BPH kosztują 560 zł). Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. Odgrażają się też polskie banki. | Pracę może stracić 10 tys. osób. 500 zł, na giełdzie papiery BPH kosztują 560 zł). Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. Odgrażają się też polskie banki. |
| **TEST2** | Pracę może stracić 10 tys. Co ważniejsze, stanie się niekwestionowanym liderem w Europie Środkowej. Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Oba banki rozmawiały dwa lata temu o fuzji z HVB, ale nie doszły do porozumienia. Odgrażają się też polskie banki. | Pracę może stracić 10 tys. osób. Co ważniejsze, stanie się niekwestionowanym liderem w Europie Środkowej. Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. Odgrażają się też polskie banki. | Pracę może stracić 10 tys. osób. Co ważniejsze, stanie się niekwestionowanym liderem w Europie Środkowej. Dotychczasowy szef HVB Dieter Rampl zostanie zaś szefem rady nadzorczej. Miliard euro do wzięcia. Odgrażają się też polskie banki. |

## Dokad zmierza chinska nauka

| | P1 | P2 | P3 |
|---|---|---|---|
| **TEST0** | Planowałem nauczyć się chińskiego i podjąć pracę, ale Chiny mnie wciągnęły. Trudno było nauczyć się chińskiego? Na dokładkę Chiny mają mnóstwo dialektów, gwar itd. Czy w Pekinie jest wielu naukowców cudzoziemców? | Trudno było nauczyć się chińskiego? Na dokładkę Chiny mają mnóstwo dialektów, gwar itd. Czy w Pekinie jest wielu naukowców cudzoziemców? Teraz jest 5 tys. cudzoziemców, ale połowa z nich przyjechała z Zachodu. Czy Chiny dościgną kiedyś USA pod względem naukowym? | Trudno było nauczyć się chińskiego? Na dokładkę Chiny mają mnóstwo dialektów, gwar itd. Czy w Pekinie jest wielu naukowców cudzoziemców? Teraz jest 5 tys. cudzoziemców, ale połowa z nich przyjechała z Zachodu. Czy Chiny dościgną kiedyś USA pod względem naukowym? |
| **TEST1** | Jak trafiłeś do Chin? Czy w Pekinie jest wielu naukowców cudzoziemców? A co będziesz robił dalej? W Chinach nie ma habilitacji. Praca dla tłumacza zawsze się znajdzie. | Jak trafiłeś do Chin? Trudno było nauczyć się chińskiego? A co będziesz robił dalej? W Chinach nie ma habilitacji. Praca dla tłumacza zawsze się znajdzie. | Jak trafiłeś do Chin? Czy w Pekinie jest wielu naukowców cudzoziemców? A co będziesz robił dalej? W Chinach nie ma habilitacji. Praca dla tłumacza zawsze się znajdzie. |
| **TEST2** | Jak trafiłeś do Chin? Czy w Pekinie jest wielu naukowców cudzoziemców? A co będziesz robił dalej? W Chinach nie ma habilitacji. Praca dla tłumacza zawsze się znajdzie. | Jak trafiłeś do Chin? Trudno było nauczyć się chińskiego? A co będziesz robił dalej? W Chinach nie ma habilitacji. Praca dla tłumacza zawsze się znajdzie. | Jak trafiłeś do Chin? Trudno było nauczyć się chińskiego? A co będziesz robił dalej? W Chinach nie ma habilitacji. Praca dla tłumacza zawsze się znajdzie. |

## Mlodzi w Polsce przestaja byc bierni i bezradni

| | P1 | P2 | P3 |
|---|---|---|---|
| **TEST0** | Gdy trzy lata temu przedstawialiśmy portret młodych 2002, nadaliśmy mu tytuł" Młodzi za burtą". Dziś sytuacja się zupełnie zmieniła. Jednak przede wszystkim zmienili się sami młodzi. Mimo że obiektywnie sytuacja na rynku pracy niewiele się zmieniła, dzisiejsza młodzież nauczyła się wykorzystywać okazje, jest znacznie bardziej dynamiczna i aktywna- komentuje Tomasz Karoń. Młodzi chwytają się wielu zajęć, doszkalają, częściej studiują- wszystko, by zwiększyć swoje szanse na rynku pracy. | Gdy trzy lata temu przedstawialiśmy portret młodych 2002, nadaliśmy mu tytuł" Młodzi za burtą". Z 27 do 18 proc. spadła też liczba młodych, którzy nic nie robią- wynika z tegorocznego raportu" Młodzi 2005" wykonanego na zamówienie AIG OFE i" Gazety". Mimo że obiektywnie sytuacja na rynku pracy niewiele się zmieniła, dzisiejsza młodzież nauczyła się wykorzystywać okazje, jest znacznie bardziej dynamiczna i aktywna- komentuje Tomasz Karoń. Młodzi chwytają się wielu zajęć, doszkalają, częściej studiują- wszystko, by zwiększyć swoje szanse na rynku pracy. | Gdy trzy lata temu przedstawialiśmy portret młodych 2002, nadaliśmy mu tytuł" Młodzi za burtą". Dziś sytuacja się zupełnie zmieniła. Uczy się i pracuje 24 proc. Mimo że obiektywnie sytuacja na rynku pracy niewiele się zmieniła, dzisiejsza młodzież nauczyła się wykorzystywać okazje, jest znacznie bardziej dynamiczna i aktywna- komentuje Tomasz Karoń. Młodzi chwytają się wielu zajęć, doszkalają, częściej studiują- wszystko, by zwiększyć swoje szanse na rynku pracy. |
| **TEST1** | Dwa razy więcej jednocześnie pracuje i się uczy. Dziś sytuacja się zupełnie zmieniła. Uczy się i pracuje 24 proc. Jednak przede wszystkim zmienili się sami młodzi. Prof. Szafraniec podkreśla rolę edukacji. | Dwa razy więcej jednocześnie pracuje i się uczy. Dziś sytuacja się zupełnie zmieniła. Uczy się i pracuje 24 proc. Jednak przede wszystkim zmienili się sami młodzi. Prof. Szafraniec podkreśla rolę edukacji. | Dwa razy więcej jednocześnie pracuje i się uczy. Dziś sytuacja się zupełnie zmieniła. Uczy się i pracuje 24 proc. Jednak przede wszystkim zmienili się sami młodzi. Prof. Szafraniec podkreśla rolę edukacji. |
| **TEST2** | Dwa razy więcej jednocześnie pracuje i się uczy. Uczy się i pracuje 24 proc. Jednak przede wszystkim zmienili się sami młodzi. Wzorzec szybkich karier z lat 90. Prof. Szafraniec podkreśla rolę edukacji. | Dwa razy więcej jednocześnie pracuje i się uczy. Dziś sytuacja się zupełnie zmieniła. Uczy się i pracuje 24 proc. Wzorzec szybkich karier z lat 90. Prof. Szafraniec podkreśla rolę edukacji. | Dwa razy więcej jednocześnie pracuje i się uczy. Dziś sytuacja się zupełnie zmieniła. Uczy się i pracuje 24 proc. Wzorzec szybkich karier z lat 90. Prof. Szafraniec podkreśla rolę edukacji. |

## Pierwsze dziecko z przeszczepu jajnika innej kobiety

| | P1 | P2 | P3 |
|---|---|---|---|
| **TEST0** | Melanie dochowała się trzech zdrowych córek. Cierpiąca na przedwczesną niewydolność jajników Stephanie przeszła menopauzę już w 14. Pomóc jej próbowała również Melanie. Louis kierowany przez dr Shermana Silbera pobrał jeden z jajników Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować. | Melanie dochowała się trzech zdrowych córek. Cierpiąca na przedwczesną niewydolność jajników Stephanie przeszła menopauzę już w 14. Pomóc jej próbowała również Melanie. Louis kierowany przez dr Shermana Silbera pobrał jeden z jajników Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować | Melanie dochowała się trzech zdrowych córek. Cierpiąca na przedwczesną niewydolność jajników Stephanie przeszła menopauzę już w 14. Pomóc jej próbowała również Melanie. Louis kierowany przez dr Shermana Silbera pobrał jeden z jajników Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować. |
| **TEST1** | Siostra pomaga siostrze. Melanie dochowała się trzech zdrowych córek. Pomóc jej próbowała również Melanie. Szansa dla innych? Już po trzech miesiącach Stephanie zaczęła miesiączkować. | Siostra pomaga siostrze. Melanie dochowała się trzech zdrowych córek. Pomóc jej próbowała również Melanie. Szansa dla innych? Już po trzech miesiącach Stephanie zaczęła miesiączkować. | Siostra pomaga siostrze. Melanie dochowała się trzech zdrowych córek. Pomóc jej próbowała również Melanie. Szansa dla innych? Już po trzech miesiącach Stephanie zaczęła miesiączkować. |
| **TEST2** | Ich życie potoczyło się jednak całkiem odmiennie. Melanie dochowała się trzech zdrowych córek. Jej siostra nie miała tego szczęścia. Pomóc jej próbowała również Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować. | Siostra pomaga siostrze. Melanie dochowała się trzech zdrowych córek. Jej siostra nie miała tego szczęścia. Pomóc jej próbowała również Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować. | Siostra pomaga siostrze. Melanie dochowała się trzech zdrowych córek. Jej siostra nie miała tego szczęścia. Pomóc jej próbowała również Melanie. Już po trzech miesiącach Stephanie zaczęła miesiączkować |

# 9. Załącznik D: Opis danych testowych

## *9.1. Zbiór Onet*

Zbiór zawiera 451 dokumentów tekstowych podzielonych na 10 kategorii:

- Ciekawostki
- Gospodarka
- Internet
- Komputery
- Kraj
- Kultura
- Nauka
- Pop-kultura
- Sport
- Świat

Dokumenty zostały przygotowane z artykułów prasowych opublikowanych na portalu www.onet.pl.

# 10. Literatura

[Abney, 1991] S. Abney: *Parsing by Chunks*, In Robert Berwick, S. A. And Tenny, C., editors, *Principle-Based Parsing*. Kluwer Academic Publishers, Dordrecht, 1991

[Agrawal, 1996] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo: *Fast discovery of association rules*, In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, Ramasamy Uthurusamy (Eds.), Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, Melo Park, CA, 1996.

[Agrawal, Imielinski, Swami, 1993]] Agrawal R., Imielinski T., Swami A., *Mining Associations Rules between Sets of Items in Large Databases*; Proc. of the ACM SIGMOD Conf. on Management of Data. Washington DC 1993.

[Agrawal, Srikant, 1994] Agrawal R., Srikant R., *Fast Algorithms for Mining Association Rules in Large Databases*, Int'l Conf. on VLDs; Santiago, Chile, 1994.

[Agrawal, Srikant, 1995] Agrawal R., Srikant R.: Mining Association Rules. Proc. of the 21st VLDB Conf. Zurich Switzerland 1995

[Banerjee, 1998] Banerjee A., Davison D., Hirsh H., Macskassy S., *Human Performance on Clustering Web Pages*, Proceedings of KDD'98 conf., 1998

[Barzilay, Elhadad, 1997] R. Barzilay, M. Elhadad: *Using Lexical Chains for Text Summarization*. Mathematic and Computer Science Dept. Ben Gurion University in the Negev, Izrael, 1997

[Bayardo, Agrawal, 1999] Bayardo R.J. Jr., Agrawal R., *Mining the Most Interesting Rules*; ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining1999.

[Bikel, 1997] D. Bikel, S. Miller, R. Szwartz, R. Weischedel: *Nymble: A High-performance Learning Name-finder*. In Proceedings of ANLP-1997, Washington DC, strony 195-201, 1997.

[Borthwick, 1999] A. Borthwick: *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. (1999) New York University, Department of Computer Science, Courant Institute, 1999

[Brants, 2000] T. Brants: *TnT – A statistical Part-of-speech Tagger*. In Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000, Seatle, USA, 2000

[Brill, 92] E. Brill: *A SimpleRule-Based Part-of-Speech Tagger*. In Proceedings of the Third Conference on Applied Computational Linguistics (ACL), Trento, Italy, 1992

[Church, 1988] K. Church: *A stochastic part program and noun phrase parser for unrestricted text*. In Proceedings of Second Conference on Applied Natural Language Processing, Austn, Texas, USA, strony 136-143, 1998.

[Ciura, Grund, Kulików, 2004] Ciura M., Grund D., Kulików S., Suszczanska N., *A System to Adapt Techniques of Text Summarizing to Polish*. International Conference on Computational Intelligence 2004: 117-120, 2004.

[Coates-Stephens, 1992] S. Coates-Stephens: *The Analysis and Acquisition of Proper Names for Robust Text Understanding*. PhD Thesis, Department of Computer Science, City University London, 1992.

[Cole, Mariani, 1996] R. A. Cole, J.Mariani, H. Uszkoreit, A.Zaenen, V.Zue: *Survey of the State of the Art in Human Language Technology*, Cambridge University Press ISBN 0-521-59277-1, 1996

[DeJong, 1997] G. DeJong: *Skomming stories in real time: an experiment in integrated understanding*. Ph.D. thesis, Computer Science Department, Yale University, 1997.

[Edmunson, 1969] H. P. Edmunson: *New methods in automatic abstracting*. Journal of the ACM, strony 264-285, 1969

[Endres-Niggemeyer, 1998] Endres-Niggemeyer, B. *Summarizing Information.* Springer, New York, NY, 1998.

[Endres-Niggermeyer, 1999] Brigitte Endres-Niggemeyer: *Two-stage cognitive modeling for human-style summarizing*, Fachhochschule Hannover, University for Applied Sciences, Department of Information and Communication, Hanover, 1999

[Ester, 1996] M. Ester, H. P. Kriegel, J. Sander, X. Xu, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise*, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining. Portland, 226-231, 1996

[Gajecki, 2004] Gajecki M., *Automatic Text Clustering in the Polish Language*. Intelligent Information Systems 2004: 419-423, 2004.

[Gallippi, 1996] A. F. Gallippi: *Learning to Recognize Names Across Languages*. In Proceedings of the Sixteenth International Conference on Computational Linguistics, Denmark, 1996

[Ganapathiraju, 2002] Ganapathiraju M. K., Relevance of Cluster size in MMR based Summarizer, A Report, Self-paced lab in Information Retrieval

[Gawrysiak, 1999] Gawrysiak P., *Using Data Mining Methodology for Text Retrieval*, International Information Science and Education Conference proceedings, DIBS99, 1999.

[Gawrysiak, 2000] Gawrysiak P., *Towards Intelligent Internet Search Systems*, 2nd Multimedia and Networked Information Systems Conference, MISSI'2000, 2000.

[Gawrysiak, 2001] P. Gawrysiak: *Automatyczna kategoryzacja dokumentów*, PhD thesis, Politechnika Warszawska, 2001

[Gawrysiak, Okoniewski, 2001] Gawrysiak P., Okoniewski M., *Knowledge Discovery in Internet*, Archiwum Informatyki Teoretycznej i Stosowanej, vol.12(2000), p.203-233, 2001.

[Gawrysiak, Rybiński, Gajda, 2004] Gawrysiak P., Rybinski H., Gajda D., Golebski M., *Extending open source software solutions for CRM text mining*, ICWI 2004: 869-872.

[Gawrysiak, Rybiński, Okoniewski, 2004] Gawrysiak P., Rybiński H., Okoniewski M., *Dynamic KOS building & management for library information systems*, Presentation from SEMKOS project, 2004

[Gillam, 1999] R. Gillam: *Text Boundary Analysis in Java*. IBM Corp., Strona WWW: http://www.ibm.com/java/education/boundaries/boundaries.html, 1999

[Goldstein, 1999] J. Goldstein: *Automatic TextSummarization of Multiple Documents*. Thesis proposal, Language Technologies Institute, Carnegie Mellon University, Pittsburg, 1999

[Goldstein, Kantrowitz, 1999] Goldstein J., Kantrowitz M., Mittal V., Carbonell J.: *Summarizing Text Documents: Sentence Selection and Evaluation Metrics*, http://citeseer.ist.psu.edu/556444.html, 1999

[Gray, Orlowska, 1998] B. Gray, M.E. Orlowska, *Clustering categorical attributes into interesting association rules*, Proceedings of the Second Pacific-Asia Conference on PAKDD'1998.

[Greffenstette, 2000] G. Greffenstette, A. Schiller, S. Ad t-Mokhtar: *Recognizing Lexical Patterns in Text*. In: Van Eynde, D. Gibbon (eds.): Lexicon Development for Speech and Language Processing, Kluwer Academic Publishers, 2000

[Greffenstette, Tapanainen, 1994] G. Greffenstette, P. Tapanainen: *What is a word, what is a sentence ? problems of tokenization*. In third International Conference on Computational Lexicography (Complex'94), strony 79-87, Budapeszt. Research Institute for Linguistics Hungarian Academy of Sciences, 1994

[Grover, 2000] C. Grover, C. Matheson, A. Mikheev and M. Moens: *LT TTT – A Flexible Tokenisation Tool*. In Proceedings of the COLING-2000. Association for Computational Linguistics, Morgan Kaufmann, 2000

[Guo, 2003] Guo G., Wang H., Bell D., Bi Y., Greer K., *Using kNN Model-based Approach for Automatic Text*, Submitted to Soft Computing Journal, October, 2003

[Guo, 2003a] Guo G., Wang H., Bell D., Bi Y., Greer K., *KNN Model-Based Approach in Classification*, In Proc.: International Conference on Ontologies, Databases and Applications of Semantics, ODBASE 2003, Catania, Sicily (Italy), 3-7 November 2003. Spinger-Verlag LNCS/AI.

[Guo, 2004] Guo G., Wang H., Bell D., Bi Y., Greer K., *An kNN Model-based Approach and its Application in Text Categorization*, Proc. of 5th International

Conference on Intelligent Text Processing and Computational Linguistic, CICLing-2004, LNCS 2945, Springer-Verlag, pages 559-570.

[Hartigan, 1979] Hartigan J., Wong M., *A k-means clustering algorithm*, Applied Statistics, 1979

[Hassel, 2004] Hassel J., *Automatic Text Summarization*, NADA-IPLab, Kungliga Tekniska Hogskolan, 2004

[Hinneburg, 1999] Hinneburg A., Keim D. A., *Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering*, Proceedings of the 25th VLDB Conference,Edinburgh, Scotland, 1999.

[Hovy, 2000] Hovy, E. and Marcu, D., *Automated Text Summarization Tutorial.* Proceedings of the Thirty- sixth Conference of the Association of Computational Linguistics (ACL-98). Pre-conference tutorial, 2000.

[Jacobs, Rau, 1990] P. Jacobs, L. Rau: *SCISOR: Extracting information from on-line news*. Communications of the ACM, 33, strony 88-97.

[Joachims, 1996] Joachims T. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*, Proceedings of ICML-97, 14th International Conference on Machine Learning, 1996

[Kierski, Okoniewski, Gawrysiak, 2003] Kierski P., Okoniewski M., Gawrysiak P., *Automatic Classification of Executable Code for Computer Virus Detection*, IIS 2003: 277-284.

[Klopotek, 2005] Kłopotek M., *Very large Bayesian multinets for text classification*, Future Generation Comp. Syst. 21(7): 1068-1082, 2005.

[Kłopotek, Wierzchom, Michalewicz, 2001] Kłopotek M., Wierzchom S., Michalewicz M., Bednarczyk M., Pawłowski W., Wasowski A., *Bayesian Network Mining System*. Intelligent Information Systems 2001: 179-193, 2001.

[Klopotek, Wierzchon, Trojanowski, 2004] Kłopotek M., Wierzchoń S., Trojanowski K.: *Intelligent Information Processing and Web Mining*, Proceedings of the International IIS: IIPWM'04 Conference held in Zakopane, Poland, May 17-20, 2004.

[Królikowski, Morzy, Perek, 2003] Królikowski Z., Morzy M., Perek J., *Set-Oriented Indexes for Data Mining Queries*, ICEIS (2) 2003: 316-323.

[Kryszkiewicz, 1998] Kryszkiewicz M., *Representative Association Rules*; Proc. of PAKDD 1998 Melbourne, Australia

[Kryszkiewicz, 1998] Kryszkiewicz M., *Representative Association Rules and Minimum Condition Maximum Consequence Association Rules*; Proc. of PAKDD 1998

[Kryszkiewicz, 2000] Kryszkiewicz M., *Mining with Cover and Extension Operators*, Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, 2000.

[Kryszkiewicz, 2001] Kryszkiewicz M., *Concise Representation of Frequent Patterns Based on Disjunction-Free Generators*, Proceedings of the 2001 IEEE International Conference on Data Mining, 2001.

[Kryszkiewicz, Rybiński, 1999] Kryszkiewicz M., Rybiński H., *Incomplete Database Issues for Representative Association Rules*, Proceedings of the 11th International Symposium on Foundations of Intelligent Systems, 1999.

[Kryszkiewicz, Rybiński, Gajek, 2004] Kryszkiewicz M., Rybiński H., Gajek M., *Dataless Transitions Between Concise Representations of Frequent Patterns*, Journal of Intelligent Information Systems, 2004.

[Kupiec, 1992] J. Kupiec: *Robust part-of-speech tagging using a hidden Markov model*. Computer Speech and Language, 6, strony 225-242, 1992

[Kupiec, 1995] J. M. Kupiec, J. Pederson, F. Chen: *A trainable document summarizer*. In Proceedings of 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, strony 68-73, Seattle, Washington.

[Levine, 1992] J. Levine, T. Mason, D. Brown, 1992: *Lex & Yacc*. O'Reilly & Associates Inc. 1992

[Lucia, 2003] Lucia H., Machado R., Thiago A., Salgueiro P., Celso A.: *A Comparison of Automatic Summarizers*, http://citeseer.ist.psu.edu/656975.html, 2003

[Luhn 1958] H. P. Luhn: *The automatic creation of literature abstracts*. IBM Journal, strony 159-165, 1958.

[Mandelbrot, 1954] Mandelbrot B., *Structure formelle des textes et communication*, Word, vol. 10, p.1-27, 1954

[McKeown, Radev, 1995] Kathleen McKeown, Dragomir R. Radev: *Generating Summaries of Multiple News Articles*, 1995

[Mikheev, 2000] A. Mikheev: *Tagging Sentence Boundaries*. Technical report, University of Edinburgh, 2000

[Milenova, 2002] Milenova B. L., Campus M. M., *O-Cluster: Scalable Clustering of Large High Dimensional Data Set*, In Proc. 2002 IEEE Int. Conf. On Data Mining (ICDM'02), 290-297, 2002

[Morfeusz] http://nlp.ipipan.waw.pl/~wolinski/morfeusz/morfeusz.html

[Morzy, Morzy, Królikowski, 2004] Morzy M., Morzy T., Królikowski Z.: *Incremental Association Rule Mining Using Materialized Data Mining Views*, ADVIS 2004: 77-87.

[Muraszkiewicz, 2002] Muraszkiewicz M., *Mobile Society, Technology, and Culture,* WITASI, 2002: 187-198.

[Muraszkiewicz, Rybiński, 1993] Muraszkiewicz M., Rybinski H., Struk W., *MULTHES-ISIS: A Flexible Software for Multilingual Thesaurus Building*. Terminology and Knowledge Engineering, 1993: 152-159.

[Nagao, Hasida, 1998] K. Nagao, K. Hasida: *Automatic Text Summarization Based on Global Document Annotation*, 1998

[Neto, 2000] Neto J. L., Santos A. D., Kaestner C., Freitag A.: *Generating Text Summaries through the Relative Importance of Topics,* http://citeseer.ist.psu.edu/neto00generating.html, 2000

[Neto, 2000a] Neto J. L., Santos A. D., Kaestner C., Freitag A.: *Document Clustering and Text Summarization*, http://citeseer.ist.psu.edu/laroccaneto00document.html, 2000

[Paice, 1990] C. Paice: *Constructing literature abstracts by computer*. Techniques and prospects. Information Processing and Management, 26: strony 171-186, 1990.

[Palmer, Hearst, 1994] D. Palmer, M. Hearst: *Adaptive sentence boundary disambiguation*. In Proceedings of the 4th Conference on Applied Natural Language Processing, Stuttgart, Niemcy, 1994

[Pedersen, Bruce, 1997] T. Pedersen, Rebecca Bruce: *Unsupervised Text Mining*. Technical Report 97-CSE-9, Department of Computer Science and Engineering, Southern Methodist University, Dallas, 1997.

[Peh, Ting, 1996] L. Shiuan Peh, Christopher Hian Ann Ting: *A divide-and-conquer strategy for parsing*. In Proceedings of the ACL/SIGPARSE 5th International Workshop on Parsing Technologies, strony 57-66, 1996

[Piskorski, 2001] Jakub Piskorski: *Shallow text processor for information extraction from free-text business documents*, Akademia Ekonomiczna w Poznaniu, 2001

[Piskorski, Homola, 2004] Piskorski J., Homola P., Marciniak M., Mykowiecka A., Przepiórkowski A.,Woliński M. *Information Extraction for Polish Using the SProUT Platform*, In the proceedings of Intelligent Information Systems 2004 (New Trends in Intelligent Information Processing and Web Mining).

[Porter, 1980] Porter, *An algorithm for suffix stripping*, vol. 14, ch. Program, pp. 130–137, 1980

[Preston, Williams, 1994] K. Preston, S. Williams: *Managing the information overload*. Physics in Business.

[Przepiórkowski, 2004] Przepiórkowski A. *Korpus IPI PAN. Wersja wstępna*, IPI PAN, Warszawa, 2004

[Przepiorkowski, 2004] Przepiórkowski A., *Towards the Design of a Syntactico-Semantic Lexicon for Polish*. Intelligent Information Systems 2004: 237-246

[Przepiórkowski, 2005] Przepiórkowski A. *The Potential of the IPI PAN Corpus*, 2005

[Przepiórkowski, Kupść, 2002] Przepiórkowski A., Kupść A., Marciniak M. Mykowiecka A. *Formalny opis języka polskiego: Teoria i implementacja*. Warszawa, Akademicka Oficyna Wydawnicza EXIT, 2002

[Przepiórkowski, Kynicki, 2004] Przepiórkowski A., Krynicki Z., Dębowski Ł., Woliński M., Janus D., Bański P.. *A Search Tool for Corpora with Positional Tagsets and Ambiguities*. In the Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, pp. 1235-1238, 2004

[Przepiórkowski, Woliński, 2003] Przepiórkowski A., Woliński M.. *A Flexemic Tagset for Polish*. In: The Proceedings of the Workshop on Morphological Processing of Slavic Languages, EACL 2003

[Przepiórkowski,2, 2005] Przepiórkowski A , *The IPI PAN Corpus in Numbers*, In the Proceedings of the 2nd Language & Technology Conference, Poznań, Poland, 2005

[Radev, 1997] Dragomir R. Radev: *Generating Natural Language Summaries from Multiple On-line sources*, Ph.D. thesis proposal, Department of Computer Science, Columbia University, New York, 1997.

[Reynat, Ratnaparkhi, 1997] J. Reynar, A. Ratnaparkhi: *A maximum entropy approach to identifying sentence boundaries*. In Proceedings of the 5th Conference on Applied Natural Language Processing, Washington D.C., USA, 1997

[Rocchio, 1971] Rocchio J. J. *Relevance Feedback in Information Retrieval*. In G. Salton, editor, The SMART Retrieval System: Experiments in Automatic Document Processing, pages 313--323. Prentice-Hall, Englewood Cliffs, NJ, 1971

[Sag, Wasow, 1999] I. Sag, T. Wasow: *Syntactic Theory: A Formal Introduction*. CSLI Publications, 1999

[SAIC, 1998] SAIC, editor: *Seventh Message Understanding Conference (MUC-7)*. Strona WWW http://www.muc.saic.com, 1998

[Savasere, Omiecinski, Navathe, 1995] Savasere A., Omiecinski E., Navathe S., *An Efficient Algorithm for Mining Association Rules in Large Databases*; In Proc. of the 21st Int'l Conf. on Very Large Data-Bases Zurich 1995.

[Schapire, 1998] Schapire R., Singer Y., Singhal A., *Boosting and Rocchio Applied to Text Filtering*, Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval, 1998

[Schmid, 2000] H. Schmid: *Unsupervised Learning of Period Disambiguation for Tokenization*. Internal report, IMS, University of Stuttgart, 2000

[Sholom, 2000] Sholom W., White B., Apte C., *Lightweight Document Clustering*, IBM T.J. Watsan Research Center, 2000

[Smyth, Goodman, 1991] P. Smyth and R.M. Goodman, *Rule induction using information theory*. "Knowledge Discovery in Databases", 1991.

[Sołdacki, Borycki, 2002], Sołdacki P., Borycki Ł., *Automatyczna klasyfikacja tekstów*, MISSI Conference, 2002.

[Steinbach, 2000] Steinbach M., Karypis G., Kumar V., *A Comparison of Document Clustering Techniques*, Technical Report 00-034. Department of Computer Science and Engineering, University of Minnesota, 2000

[Sundheim, 1995] B. Sundheim, editor: *Sixth Message Understanding Conference (MUC-6)*. Distributed by Morgan Kaufmann Publishers, Inc., San Mateo, California, 1995

[Suszczanska, Kulików, 2003] Suszczanska N., Kulików S.: *A Polish Document Summarizer*. Applied Informatics 2003: 369-374

[Tait, 1983] J. Tait: *Automatic summarizing of English texts*. Ph.D. thesis, University of Cambridge, Cambridge, England.

[Tkach, 1999] D. Tkach: *The pillars of knowledge management*. In Knowledge Management, 2(3), page 47, 1999

[Wakao, 1996] T. Wakao, R. Gaizauskas, Y. Wilks: *Evaluation of an Algorithm for the Recognition and Classification of Proper Names*. In Proceedings of the 16th International Conference on Computational Linguistics (COLING), strony 418-423, Dania, 1996

[Witbrock, Mittal, 1999] Michael J. Witbrock Vibhu O. Mittal: *Ultra-Summarization: A Statistical Approach to Generating Highly Condensed Non-Extractive Summaries*, 1999

[Woliński, 2003] Woliński M., *System znaczników morfosyntaktycznych w korpusie IPI PAN*, Polonica, XXII–XXIII, s. 39–55, 2003

[Woliński, Przepiórkowski, 2001] Woliński M., Przepiórkowski A., *Projekt anotacji morfosynktaktycznej korpusu języka polskiego*, Prace IPI PAN 938, Instytut Podstaw Informatyki Polskiej Akademii Nauk, 2001

[Yang, 1998] Yang Y., Liu X., *A re-examination of text categorization methods*, ACM SIGIR Conference on Research and Development in Information Retrieval, New York, 1998

[Zaki, 2000] Zaki M.J., *Generating Non-Redundant Association Rules*; KDD 2000 Boston MA USA

[Zipf, 1949] Zipf G., *Human Behaviour and the Principle of Least Effort*, Cambridge, 1949

## *10.1. Literatura uzupełniająca*

Ewa i Feliks Przyłubscy : *Język polski na co dzień*

Michał Jaworski: *Podręczna gramatyka języka polskiego*

Piotr Bąk: *Gramatyka języka polskiego*

Włodzimierz Gruszczyński, Jerzy Bralczyk: *Słownik gramatyki języka polskiego*

# 11. Projekty w ramach prac badawczych

Niniejsze projekty zostały wykonane pod kierownictwem autora niniejszej rozprawy w ramach przedmiotu „Inteligentne Systemy Informatyczne" prowadzonego przez prof. dr hab. inż. Mieczysława Muraszkiewicza na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej.

[Proj1] Dąbrowski M.: *Tokenizacja i podział na zdania*, ISI, Java 1.4, 2004

[Proj2] Grula M.: *Tokenizacja i podział na zdania – podejście algorytmiczne*, ISI, 2005

[Proj3] Zagniński P.: *Tokenizacja i podział na zdania w oparciu o wyrażenia regularne*, ISI, C# .NET 2.0, 2005

[Proj4] Lipiński T., Niewiadomski H., Pleban M., Portalski M., Rządca K., Tucholski P.: *Polski stemmer*, ISI, Visual C++, 2002

[Proj5] Bień A., Rzońca K.: *Rozpoznawanie części mowy*, ISI, Java 1.4, 2005

[Proj6] Juchnowicz M.: *Rozpoznawanie części mowy*, ISI, C# .NET 2.0, 2005

[Proj7] Dziedziniewicz K., Obrycki L.: *Rozpoznawanie zaimków i nazw własnych*, ISI, C# .NET 1.0, 2005

[Proj8] Karpiński M., Trybalski P.: *Usuwanie niejednoznaczności*, ISI, Java 1.4, 2004

[Proj9] Olesiński A., Zubka A.: *Rozkład zdań złożonych na zdania proste*, ISI, Java 1.4, 2004

[Proj10] Korzyński A.: *Rozkład zdań złożonych na zdania proste*, ISI, C# .NET 1.0, 2005

[Proj11] Rama G., Zdeb K.: *Rekurencyjny rozbiór zdań*, ISI, Java 1.4, 2004

[Proj12] Bartosiewicz P., Różyk D.: *Rekurencyjny rozbiór zdań*, ISI, Java 1.4, 2004

[Proj13] Zarzycki P., Pawłowski N.: *Adaptacyjny rozbiór zdań*, ISI, Java 1.4, 2004

[Proj14] Reszka Ł., Warda M.: *Relacyjny rozbiór zdań*, ISI, Java 1.4, 2004

[Proj15] Danilewicz P., Miącz M.: *Relacyjny rozbiór zdań*, ISI, C# .NET 1.1, 2005

[Proj16] Kaczyński P., Królak P.: *Sumaryzacja dokumentów tekstowych*, ISI, C# .NET 2.0, 2005

# 12. Indeks pojęć polskich

# 13. Indeks pojęć angielskich

# 14. Indeks skrótów