Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

Rok akademicki 2004/2005

PRACA DYPLOMOWA MAGISTERSKA

Michał Kosmulski

Reprezentacja dokumentów tekstowych w modelu przestrzeni wektorowej

Opiekun pracy
dr inż. Piotr Gawrysiak

Ocena .....

.....
Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Specjalność Inżynieria oprogramowania i systemy informacyjne

Data urodzenia

Data rozpoczęcia studiów

Życiorys

Zyciorys usuniety dla ochrony prywatnosci

.....

podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu .....2005 r.

z wynikiem .....

Ogólny wynik studiów .....

Dodatkowe wnioski i uwagi Komisji .....

.....

.....

Streszczenie

Model przestrzeni wektorowej należy do najpopularniejszych sposobów reprezentacji dokumentów tekstowych dla celów klasteryzacji. W pracy przedstawiono algorytmy stosowane na wszystkich etapach analizy zbioru dokumentów, od wstępnego przetwarzania tekstu (rozpoznawanie języka, stemming itd.) i generacji współrzędnych wektorów aż do właściwej klasteryzacji i nadawania nazw znalezionym klastrom. Zaproponowano również dwa nowe algorytmy: SoundexPL dla uproszczonego stemmingu wyrazów w języku polskim oraz SAI dla redukcji wymiaru przestrzeni cech. Stworzona w ramach pracy aplikacja pozwala dzięki elastycznemu systemowi wtyczek na porównywanie wyników działania różnych kombinacji omówionych algorytmów.

Słowa kluczowe: VSM, model przestrzeni wektorowej, klasteryzacja, klasyfikacja tekstów, IR

Representing text documents in the Vector Space Model

The Vector Space Model belongs to the most common ways of representing text documents for the purpose of clustering. In this thesis algorithms used at all stages of analysing a set of documents are introduced, from text preprocessing (language detection, stemming etc.) and vector coordinate generation to clustering and cluster naming. Two new algorithms are suggested: SoundexPL for simplified stemming of Polish words and SAI for reducing the dimensionality of feature space. An application developed in the course of writing the thesis makes comparing results of different algorithm combinations possible thanks to a flexible plug-in system.

Key words: VSM, Vector Space Model, clustering, text classification, IR

Spis treści

Wstęp

Przedmiot i zakres pracy

Informacja przechowywana w dokumentach tekstowych stanowi ważne źródło wiedzy dostępnej bezpośrednio dla człowieka, ale trudnej do wykorzystania przez komputer. Wyszukiwanie, grupowanie i klasyfikacja dokumentów tekstowych wymagają zatem przekształcenia ich na postać nadającą się do bezpośredniego przetwarzania przez maszynę. Często stosowanym rozwiązaniem jest utworzenie wektorów nad ciałem liczb rzeczywistych reprezentujących analizowane dokumenty i wykonywanie dalszych przekształceń już tylko na tych wektorach. Podejście to nosi nazwę modelu przestrzeni wektorowej (*Vector Space Model*, VSM). Dla zagadnień wymagających porównywania ze sobą dokumentów, na przykład grupowania, wykorzystywane są wektory należące do przestrzeni metrycznych, w których zdefiniowane jest pojęcie odległości, stanowiącej zwykle miarę różnic w tematyce dokumentów.

Niniejsza praca poświęcona jest zagadnieniu reprezentacji dokumentów w przestrzeniach wektorowych, ze szczególnym uwzględnieniem przestrzeni metrycznych. Na przykładzie zagadnienia grupowania dokumentów omówione w niej zostały podstawy modelu przestrzeni wektorowej (rozdział pierwszy), główne zagadnienia związane ze wstępnym przetwarzaniem dokumentów (rozdział drugi), generacją wektorów reprezentujących dokumenty (rozdział trzeci) oraz procesem grupowania wektorów (rozdział czwarty). W pracy zaproponowano również nowe podejście do problemu stemmingu dla języka polskiego, polegające na wykorzystaniu w tym celu algorytmu opartego na algorytmie Soundex, oraz metodę SAI redukcji wymiaru przestrzeni cech.

Stworzona w ramach pracy aplikacja stanowi jedną z trzech głównych części większego systemu przeznaczonego do grupowania dokumentów wyszukiwanych w sieci WWW. Pozostałe moduły stanowią przedmiot prac magisterskich Michała Pawluczuka (pobieranie dokumentów z sieci) oraz Grzegorza Kolendo (grupowanie wektorów reprezentujących dokumenty). Wszystkie moduły zostały zaprojektowane w taki sposób, by stanowiły jak najbardziej niezależne od siebie elementy systemu i każdy mógł być wykorzystany niezależnie od innych. W niniejszej pracy proces grupowania wykorzystano do porównywania między sobą wektorów reprezentujących dokumenty tekstowe, gdyż bezpośrednia ocena poprawności sposobu generacji wektorów jest bardzo trudna ze względu na ich znaczną długość. Znajdowany przez klasteryzator zbiór

grup dokumentów umożliwia pośrednio ocenę sposobu generacji wektorów i porównanie różnych algorytmów realizujących to zadanie. W niniejszej pracy grupowanie pełni zatem jedynie rolę pomocniczą i zostało przedstawione tylko w zarysie. Kod aplikacji wykorzystuje klasy klasteryzatorów autorstwa Grzegorza Kolendo.

Jako przykładowy proces ilustrujący działanie algorytmów reprezentacji dokumentów tekstowych w modelu przestrzeni wektorowej wybrane zostało właśnie grupowanie, ponieważ w zagadnieniu tym zbiór dokumentów stanowi jedyny parametr wejściowy. Podczas gdy np. w problemie wyszukiwania haseł pojawiają się nie tylko przeszukiwane dokumenty, ale również sformułowane przez człowieka zapytanie, w procesie grupowania nie ma oprócz zbioru dokumentów żadnych danych wejściowych, których wybór dokonywany byłby przez człowieka. Dzięki temu wyniki są zależne przede wszystkim od zastosowanych sposobów reprezentacji dokumentów i metod wyznaczania ich podobieństwa, zaś wpływ pozostałych czynników jest mniejszy niż w większości pokrewnych zagadnień.

Część praktyczna pracy zaowocowała stworzeniem elastycznej aplikacji pozwalającej na porównywanie różnych metod grupowania dokumentów. Dzięki systemowi wtyczek, możliwy jest wybór dowolnej kombinacji algorytmów stosowanych na poszczególnych etapach obliczeń. Możliwe jest także rozszerzenie programu o obsługę algorytmów dotychczas w nim nie zaimplementowanych. Przy tworzeniu programu największy nacisk położono na elastyczność i możliwość wykorzystania w nim różnorodnych rozwiązań, nawet kosztem wydajności lub zwiększenia komplikacji kodu. Rozdział piąty zawiera opis stworzonej aplikacji, zaś rozdział szósty – wyniki testowania poszczególnych algorytmów za jej pomocą.

Przetwarzanie dokumentów w języku naturalnym

Przetwarzanie tekstu w językach naturalnych (*Natural Language Processing*, NLP) stanowiło zawsze jeden z podstawowych problemów w dziedzinie sztucznej inteligencji. Niektóre działy tej dziedziny, np. rozumienie i generacja mowy ludzkiej, nie spełniły dotychczas postawionych przed nimi w początkowych latach rozwoju wyolbrzymionych oczekiwań, natomiast nastąpił gwałtowny rozwój wiedzy związanej z wyszukiwaniem i ekstrakcją informacji zawartej w dokumentach tekstowych. Upowszechnienie internetu, a przede wszystkim sieci WWW jako ważnego medium komunikacyjnego i źródła informacji, nadało tej dziedzinie duże znaczenie praktyczne.

Internet nazywany bywa największą bazą danych świata, ale od większości baz danych różni się dość znacznie. Podczas gdy bazy danych przechowują informacje w postaci gotowej do użycia przez komputer, podzieloną na niewielkie fragmenty o ustalonym znaczeniu (np. krotki w relacyjnej bazie danych), dokumenty tekstowe posiadają wiele cech czyniących je źródłami informacji trudnymi do wykorzystania przez maszynę. Do cech tych należą między innymi: typowa dla języków naturalnych wysoka nadmiarowość, skomplikowana i nie zawsze jednoznaczna gramatyka oraz znaczny stopień złożoności reguł semantycznych wiążących poszczególne konstrukcje językowe z ich znaczeniami. Cechy te sprawiają, że przetwarzanie dokumentów w językach naturalnych jest znacznie bardziej skomplikowane niż ekstrakcja informacji z baz danych lub innych źródeł informacji dostosowanych do sposobu działania współczesnych komputerów.

Z drugiej strony, ilość informacji zgromadzonych w sieci czyni odnalezienie użytecznych danych niemal niemożliwym bez pomocy systemów automatyzujących ten proces. W ostatnich latach obserwuje się szybki rozwój w dziedzinach wiedzy związanych z wyszukiwaniem informacji w internecie, napędzany nie tylko przez ośrodki akademickie, ale przede wszystkim przez firmy oferujące tego rodzaju usługi. Razem z pojawieniem się i rozwojem nowoczesnych wyszukiwarek internetowych wzrosło również zainteresowanie zagadnieniami pokrewnymi wyszukiwaniu, a więc tworzeniem map zasobów sieciowych, wyszukiwaniem dokumentów podobnych do zadanego i grupowaniem dokumentów. W związku ze zwiększeniem się ilości spamu rozpowszechnianego za pomocą poczty elektronicznej, dużego znaczenia praktycznego nabrały algorytmy klasyfikacji dokumentów, pozwalające na tworzenie filtrów wykrywających automatycznie niechcianą pocztę. Wymienione zagadnienia w wielu przypadkach zazębiają się, a odpowiednie połączenie związanych z nimi technik pozwala na wygodniejszy i szybszy dostęp do potrzebnych informacji.

Potrzeba umożliwienia programom komputerowym „rozumienia” treści dokumentów dostępnych w internecie doprowadziła do powstania idei *Semantic Web* (SW). System ten ma opierać się na specjalnie przygotowanych dokumentach zawierających informacje zarówno dla ludzi, jak i dla maszyn. Już obecnie coraz częściej wykorzystuje się pewne elementy tego przyszłego standardu ([SW]), głównie znakowanie dokumentów za pomocą RDF (*Resource Description Framework*), na przykład w celu zaznaczenia informacji o prawach autorskich, oraz systemy FOAF (*Friend of a Friend*)

wykorzystywane do modelowania relacji między użytkownikami na forach dyskusyjnych i w komunikatorach internetowych. Przygotowanie specjalnie oznaczonych dokumentów nie zawsze jest jednak możliwe, tak więc pomimo rozwoju SW potrzeba przetwarzania dokumentów tekstowych nie wyposażonych w żadne zrozumiałe dla maszyny wskazówki dotyczące ich klasyfikacji nadal pozostaje aktualna.

Równocześnie z rozwojem globalnej sieci, rośnie ilość danych przechowywanych w sieciach wewnętrznych firm i innych instytucji, a nawet na komputerach osobistych poszczególnych użytkowników. Obecnie rozwijanych jest coraz więcej programów zdolnych do analizy treści dokumentów znajdujących się na pojedynczych maszynach. Narzędzie *Google Desktop Search* może zapoczątkować nowy trend w tej dziedzinie. Analizowanie treści wysyłanej poczty elektronicznej i tworzenie na jej podstawie profilu użytkownika wykorzystywanego później w celach marketingowych stało się przyczyną kontrowersji wokół usługi *Google Mail*. Prowadzone są również badania nad technologiami mającymi ułatwić użytkownikom zarządzanie plikami przechowywanymi na ich komputerach na podstawie analizy dokumentów, takich jak edytowane pliki tekstowe, wiadomości e-mail czy rozmowy za pomocą komunikatorów internetowych. Funkcje takie mają się znaleźć w przyszłych wersjach systemów Mac OS, Windows oraz wszelkich Uniksów (kombinacja Beagle/Dashboard).

1. Model przestrzeni wektorowej

Model przestrzeni wektorowej (*Vector Space Model*, VSM) stanowi formalny sposób reprezentacji dokumentów tekstowych, wykorzystywany w zagadnieniach wyszukiwania i porządkowania informacji. W modelu tym każdy dokument jest reprezentowany przez wektor należący do pewnej przestrzeni wektorowej, zwanej przestrzenią cech (*feature space*). W praktyce stosuje się przestrzenie nad ciałem liczb rzeczywistych ($\mathbb{R}^n$), a często nawet zbiory takie jak $[0, \infty)^n$, nie będące przestrzeniami wektorowymi w ścisłym tego słowa znaczeniu. Porównywanie dokumentów realizowane jest z użyciem metryk określonych w przestrzeni cech, przy czym w praktyce niekiedy stosuje się „funkcje podobieństwa" bądź „miary odległości" nie spełniające wszystkich aksjomatów metryki (przykłady przedstawiono w [Jain]).

Reprezentacja dokumentów za pomocą wektorów pozwala na wykonywanie formalnych przekształceń, które można interpretować jako pewne operacje wykonywane na rzeczywistych dokumentach. Pozwala ona także na wykorzystanie do analizy tekstów algorytmów znanych z innych dziedzin. Przykładowo, istnieje wiele algorytmów grupowania punktów w przestrzeniach metrycznych. Dzięki przedstawieniu dokumentów w postaci wektorów, algorytmy te mogą bez żadnych zmian zostać wykorzystane do grupowania tekstów. Wykorzystanie różnych metryk umożliwia porównywanie dokumentów między sobą według różnych kryteriów.

Model przestrzeni wektorowej umożliwia także wyszukiwanie dokumentów dotyczących podanego przez użytkownika hasła. W tym celu zapytanie reprezentowane jest jako wektor tej samej przestrzeni co dokumenty, a następnie znajdowane są dokumenty, których wektory są odpowiednio bliskie wektorowi zapytania w pewnej ustalonej metryce ([Dominich]). W przypadku stosowania metryki kosinusowej (patrz punkt 3.1.2), operację tę można wyrazić jako mnożenie macierzy, której wiersze stanowią wektory odpowiadające poszczególnym dokumentom, przez wektor odpowiadający zapytaniu. Szukany zbiór dokumentów najbardziej pasujących do zapytania wskazują numery tych współrzędnych wektora wynikowego, których wartości są największe.

Tworzenie wektorowej reprezentacji zbioru dokumentów składa się najczęściej z kilku etapów. Pierwszym z nich jest wstępne przetwarzanie dokumentów, którego celem jest wyeliminowanie z nich informacji nieistotnych i mogących negatywnie wpłynąć na działanie algorytmów używanych w kolejnych krokach. Dopiero tak przefil-

trowane dokumenty są analizowane w celu znalezienia odpowiedniej bazy przestrzeni cech. Często wyznaczona baza poddawana jest pewnym przekształceniom mającym na celu redukcję jej wymiaru (np. PCA, patrz punkt 3.4.2) lub poprawienie wyników późniejszych operacji. Przykład takiego przekształcenia stanowi stosowana w niektórych systemach technika LSA (*Latent Semantic Analysis*), przedstawiona w [Deerwester], umożliwiająca przezwyciężenie części problemów związanych z występowaniem w tekście synonimów i homonimów. Wektory zmodyfikowanej przestrzeni, reprezentujące dokumenty, są ostatecznie poddawane działaniu algorytmów grupowania, klasyfikacji itp., odpowiednich do potrzeb.

Pomimo stosowania rozmaitych metod redukcji wymiaru przestrzeni cech, często zachodzi konieczność operowania wektorami należącymi do przestrzeni o bardzo wysokiej liczbie wymiarów. W większości przypadków są to jednak wektory rzadkie, czyli posiadające liczne współrzędne o wartości zero. Koniecznością jest więc zwykle odpowiednia implementacja struktur danych, umożliwiająca efektywne przechowywanie wektorów i macierzy rzadkich. Ponadto, w zastosowaniach praktycznych często zachodzi konieczność przetwarzania bardzo dużych zbiorów dokumentów. Niesie to ze sobą konieczność stosowania specjalnych rozwiązań pozwalających na ominięcie ograniczeń sprzętu, zarówno dotyczących szybkości procesorów, jak i zapotrzebowania na pamięć operacyjną. W związku z tym, często stosowane są algorytmy rozproszone, działające na klastrach składających się czasem z tysięcy maszyn (np. wyszukiwarka Google korzysta obecnie z około 30 klastrów liczących do 2000 komputerów). Ze względu na taką architekturę wielu dużych systemów przetwarzających dokumenty w języku naturalnym, algorytmy wykorzystywane w tej dziedzinie powinny cechować się możliwością znacznego zrównoleglenia obliczeń.

2. Wstępne przetwarzanie (*preprocessing*)

Języki naturalne charakteryzują się wysokim stopniem redundancji, która występuje w dokumentach tekstowych na dwóch poziomach. Pierwszy poziom dotyczy samej struktury języka: jego gramatyki i semantyki, drugi zaś związany jest z ortografią, interpunkcją oraz innymi cechami tekstu utrwalonego w postaci elektronicznej lub w ogóle pisemnej. Aby formę tekstową dokumentu zamienić na bardziej zwięzłą i dającą się porównać z reprezentacjami innych dokumentów postać wektora liczb, konieczne jest usunięcie z dokumentu nadmiarowości i pozostawienie jedynie istotnych informacji. Wstępne oczyszczenie dokumentu z informacji bezużytecznych lub wręcz szkodliwie wpływających na działanie niektórych algorytmów stanowi zwykle pierwszy krok w jego analizie.

2.1. Odczyt danych z zewnętrznego źródła

Najprostsze formy przetwarzania dokumentu mają zwykle miejsce już podczas pobierania go z zewnętrznego źródła danych i tworzenia wewnętrznych struktur danych dokument ten reprezentujących. Na tym etapie konieczne jest wydobycie treści dokumentu ze sformatowanego w pewien sposób ciągu bajtów. Odkodowanie danych może w zależności od ich formatu być trywialne, jak w przypadku odczytu plików tekstowych, lub bardzo skomplikowane, np. wtedy gdy tekst jest odczytywany z pliku edytora tekstu o złożonej budowie. Dodatkowe informacje towarzyszące tekstowi, takie jak metadane pliku bądź informacje o użytym kroju i kolorze pisma, zwykle nie są zachowywane w oryginalnej postaci, lecz całkowicie lub częściowo tracone.

Podczas wczytywania dokumentu najczęściej wykrywane są granice zdań i poszczególnych wyrazów. Tym samym tracona jest informacja na temat wielokrotnych wystąpień znaków odstępu i znaków nowego wiersza – każdy taki ciąg jest zwykle traktowany jako pojedynczy separator. Również informacje o znakach przestankowych mogą albo zostać zachowane albo odrzucone po ich wykorzystaniu do wykrycia granic zdań. Wybór znaków uznawanych za separatory słów i zdań zależy od rodzaju analizowanego dokumentu. Znaki, które w dokumentach napisanych w językach naturalnych stanowią separatory niosące stosunkowo mało informacji o treści dokumentu (np. kropka), w pewnych językach sztucznych, jak języki programowania, mogą stanowić istotne wyrażenia, znaczące dla porównywania różnych dokumentów między sobą.

Pomimo niezwykłej prostoty, do najczęściej stosowanych modeli dokumentu tekstowego należy model *bag of words* (BOW). W modelu tym nie jest uwzględniana kolejność wyrazów występujących w dokumencie, a jedynie liczba wystąpień poszczególnych słów. Stosowane są też rozszerzenia podstawowego modelu polegające na przykład na wykorzystaniu n-gramów zamiast słów. Dokument w modelu BOW może być przechowywany w postaci mapy (tablicy asocjacyjnej) odwzorowującej każde słowo lub n-gram na liczbę jego wystąpień w tekście. Dostęp do takiej struktury danych jest szybszy niż każdorazowe przeglądanie liniowo uporządkowanej listy zdań bądź słów.

2.2. Zamiana na małe litery

Celem ujednolicenia tekstu i uproszczenia pracy programu na dalszych etapach analizy dokumentu, jako jeden z pierwszych kroków jego przetwarzania wykonywana jest zwykle zamiana wszystkich liter na same tylko litery małe lub same tylko litery wielkie.

Zabieg ten, chociaż pozornie bardzo prosty, wymaga pewnej uwagi przy obsłudze niektórych języków, gdyż nie zawsze relacja pomiędzy wielkimi i małymi literami jest typu „jeden do jednego”. W języku niemieckim litera `ß` (*scharfes S*) podczas zamiany na wielkie litery jest zastępowana przez dwie litery `S`, a więc np. słowo *Straße* po zamianie na wielkie litery przyjmie postać *STRASSE*, podczas gdy ta ostania po zamianie na małe litery zmieni się w *strasse*. W dokumencie może wystąpić każda z wymienionych trzech form, a program powinien uwzględnić fakt, że za każdym razem chodzi o jedno i to samo słowo. Jednym z rozwiązań może być zastosowanie słownika wyrazów bliskoznacznych, w którym umieszczone zostaną obie formy pisowni (patrz punkt 2.6), lub osobnego filtru sprowadzającego pisownię podobnych „niebezpiecznych” słów do jednej, z góry ustalonej postaci.

Ze względu na opisane powyżej trudności, może być konieczne dostosowanie pozostałych etapów wstępnego przetwarzania dokumentu do specyfiki zaimplementowanego sposobu zamiany wszystkich liter na małe lub wielkie. W omówionym wyżej przykładzie języka niemieckiego, konstrukcja stemmera (patrz punkt 2.5) będzie zależna od tego, czy wyrazy mu przekazywane będą zawierać literę `ß`, czy też zostanie ona zastąpiona podwójnym wystąpieniem litery `S`.

2.3. Rozpoznawanie języka dokumentu

Sposób działania niektórych algorytmów wstępnego przetwarzania tekstu zależy od języka dokumentu. Przykładowo, zarówno stemming (patrz punkt 2.5) jak i znajdowanie słów bliskoznacznych (patrz punkt 2.6) przebiegają w zupełnie inny sposób dla każdego języka. Z wyjątkiem sytuacji, gdy analizowane są tylko dokumenty napisane w jednym z góry znanym języku, zachodzi konieczność jego automatycznego wykrycia.

Niekiedy możliwe jest uzyskanie informacji o języku na podstawie metadanych samego dokumentu. Za przykład mogą służyć znaczniki `META` oraz atrybuty `LANG` znaczników HTML pozwalające zdefiniować odpowiednio język całego dokumentu oraz poszczególnych jego fragmentów.

W sytuacji, gdy informacja o języku nie może zostać odczytana z metadanych, konieczne jest jej odtworzenie na podstawie treści dokumentu. Jedną z najpopularniejszych i najprostszych metod jest wykorzystanie stop-listy, przedstawione w punkcie 2.4. Inna metoda polega na analizowaniu częstości wystąpień n-gramów literowych, czyli n-literowych ciągów wchodzących w skład dokumentu. Każdy z języków posiada charakterystyczny rozkład częstości występowania różnych n-gramów literowych, zatem za język dokumentu można uznać z dużym prawdopodobieństwem ten, którego statystyki występowania poszczególnych n-gramów literowych są najbardziej zbliżone do statystyk zebranych z dokumentu. Obie metody działają równie dobrze dla dłuższych dokumentów (od około 30 wyrazów wzwyż). Dla dokumentów krótszych metoda n-gramów literowych charakteryzuje się niższą stopą pomyłek.

2.4. Stop-lista

W każdym języku występują słowa, które, pomimo że występują w tekstach bardzo często, nie wpływają bezpośrednio na treść zdań, a jedynie kształtują tok wypowiedzi. Do słów takich można zaliczyć między innymi większość spójników, rodzajniki (w językach je posiadających) oraz słowa pomocnicze służące w wielu językach do budowy czasów złożonych (np. *have* w języku angielskim). Lista takich słów oraz algorytm polegający na ich usuwaniu z treści dokumentu nosi miano stop-listy.

Usuwanie słów znajdujących się na stop-liście z treści dokumentu jest zasadne z kilku powodów. Po pierwsze, ze względu na znaczną częstość występowania, słowa te

dominują we wszelkich statystykach dokumentu opartych na zliczaniu wystąpień słów. Z tego względu wprowadzają one „szum informacyjny" utrudniający wyłuskanie z dokumentu naprawdę istotnych słów i wyrażeń. Po drugie, słowa tego rodzaju nie wnoszą żadnej lub prawie żadnej informacji o treści dokumentu, zaś ich używanie w tekście jest często wymuszone przez zasady gramatyki, a nie zamysł autora. W związku z tym, liczba i miejsce ich wystąpień nie pozwalają wyciągać żadnych wniosków na temat treści dokumentu ani jego podobieństwa do innych tekstów. Trzecim powodem korzystania ze stop-listy jest oszczędność pamięci związana z usunięciem słów nie wnoszących istotnych informacji o dokumencie.

Stop-lista może również zostać wykorzystana do rozpoznawania języka dokumentu. Mając do dyspozycji stop-listy dla kilku języków można porównać częstość występowania w tekście słów z poszczególnych list i język tej, której słowa pojawiają się najczęściej, uznać za język dokumentu.

2.5. Lematyzacja i stemming

W przypadku języków takich jak język polski, posiadających złożoną fleksję, poprawność gramatyczna zdań wymaga, aby zachodziła zgodność pomiędzy formami gramatycznymi poszczególnych wyrazów wchodzących w ich skład, pomimo że zwykle zdania te są dla człowieka całkowicie zrozumiałe, nawet gdy zgodności takiej nie ma. Pomimo, że jest gramatycznie błędne, zdanie *Ala mieć kot* jest zrozumiałe równie dobrze, jak jego poprawna forma. Odmiana czasowników przez osoby i rzeczowników przez przypadki stanowi w nim informację nadmiarową, bez której zrozumienie tego zdania wciąż jest możliwe, a w pewnych sytuacjach może nawet być łatwiejsze niż wtedy, gdy odmiana jest obecna. Osoba nie władająca językiem polskim, lecz posiadająca odpowiedni słownik, z łatwością znajdzie w nim słowo *mieć*, podczas gdy słowo *ma* może w nim nie być obecne.

Aby ułatwić porównywanie dokumentów tekstowych, w których te same słowa występują w różnych postaciach gramatycznych, wykorzystywane są algorytmy lematyzacji i stemmingu. Polegają one na sprowadzaniu słów do ich form podstawowych, co umożliwia późniejsze odnalezienie wystąpień każdego z nich w innych miejscach dokumentu, nawet jeśli występuje tam ono w odmiennych formach. Czasami do wspólnej postaci sprowadzane są także różne słowa pochodzące od tego samego rdzenia (a więc „dotyczące tego samego tematu").

Różnica pomiędzy oboma klasami algorytmów dotyczy zakresu informacji wykorzystywanych przez nie przy sprowadzaniu słowa do jego formy podstawowej. Lematyzacja uwzględnia kontekst słowa i budowę gramatyczną zdania, w którym ono występuje, aby odtworzyć jego lemat, czyli podstawową formę gramatyczną, np. mianownik liczby pojedynczej dla rzeczownika, bezokolicznik dla czasownika itd. Stemming wykorzystuje w analogicznym celu jedynie samo brzmienie słowa i często jako jego rdzeń (ang. *stem*) zwraca łańcuch nie stanowiący poprawnej formy gramatycznej, za to, przynajmniej w teorii, taki sam dla wszystkich form gramatycznych danego słowa. Z powyższego opisu widać, że lematyzacja jest procesem wprawdzie dającym lepsze wyniki (np. w sytuacji występowania homonimii), ale za to znacznie bardziej złożonym niż stemming.

Zarówno w przypadku lematyzacji, jak i stemmingu, można wyodrębnić dwa główne podejścia do problemu. Pierwsze z nich, podejście słownikowe, polega na wykorzystaniu słownika zawierającego znaczną liczbę różnych form gramatycznych poszczególnych słów oraz odpowiadającą każdej z nich formę podstawową (lemat lub rdzeń). Rozwiązanie takie pozwala wprawdzie bezbłędnie znajdować rdzeń słów zawartych w słowniku, ale za to nie dostarcza żadnych wyników w sytuacji, gdy słowa (lub jego formy użytej w dokumencie) w wykorzystywanym słowniku brakuje. Wiąże się ono również z potrzebą tworzenia i przechowywania w pamięci oraz szybkiego przeszukiwania słownika o znacznych rozmiarach. Efektywne zaimplementowanie odpowiednich struktur danych może stanowić poważną trudność.

Drugie podejście, algorytmiczne, polega na wykorzystaniu zbioru reguł pozwalających wykryć i usunąć różnice pomiędzy poszczególnymi formami gramatycznymi danego słowa czy też różnymi słowami o tym samym rdzeniu. W odróżnieniu od metod słownikowych, metody algorytmiczne nie dostarczają nigdy wyników o stuprocentowej dokładności. Zrównoważenie prostoty algorytmu z jednej, a liczby pomyłek z drugiej strony, stanowi niezbędny kompromis. Trudność stworzenia efektywnego algorytmu stemmingu bądź lematyzacji zależy od struktury języka, którego algorytm ma dotyczyć. W przypadku języka angielskiego, posiadającego ubogą fleksję i raczej proste zasady tworzenia słów pochodnych, dostępnych jest wiele algorytmicznych sposobów stemmingu i lematyzacji, które pod względem prędkości działania i wykorzystania pamięci posiadają nad metodami słownikowymi przewagę równoważącą dla większości zastosowań koszt związany z nieco wyższą stopą błędów.

Naturalnym rozwiązaniem dla pewnych zadań może okazać się użycie metody mieszanej, czyli korzystanie z podejścia słownikowego, ale z możliwością podparcia się jedną z metod algorytmicznych w przypadku braku słowa w słowniku.

2.5.1. Algorytm Lovins

Algorytm Lovins uchodzi za pierwszy opisany algorytm stemmingu dla języka angielskiego ([Porter, The Lovins stemmer]). Dzięki zastosowaniu niezwykle obszernej listy końcówek (294 pozycje), stemming jest wykonywany w zaledwie dwóch krokach i realizowany jest szybciej niż w wielu innych algorytmach.

Końcówki (*endings*) są pogrupowane według ich długości – najdłuższe z nich liczą 11 znaków, najkrótsze zaś składają się z jednej litery. Z każdą z nich związany jest jeden z 29 warunków (*conditions*), oznaczonych literami A-Z oraz AA, BB, CC. Warunki te dotyczą rdzenia słowa, a więc słowa powstającego po obcięciu końcówki rozpatrywanej w chwili sprawdzania warunku. Większość z nich służy sprawdzeniu, czy rdzeń posiada pewną minimalną długość (wyrażoną w znakach) oraz czy kończy się pewną kombinacją liter.

Pierwszy krok algorytmu polega na rozpatrywaniu końcówek wyrazu, począwszy od najdłuższych, 11-literowych. W sytuacji, gdy żadna ze znanych końcówek danej długości nie stanowi zakończenia badanego słowa, rozpatrywane są końcówki o długości o jeden mniejszej. W chwili znalezienia pasującej końcówki, sprawdzane jest zachodzenie skojarzonego z nią warunku. Jeżeli warunek zachodzi, końcówka jest usuwana ze słowa.

W drugim i ostatnim kroku, do wyniku działania kroku pierwszego stosowanych jest 35 zasad transformacyjnych (*transformation rules*). Pierwsza z nich zastępuje podwójne wystąpienia liter takich jak np. `b`, `d` i `l` na końcu słowa pojedynczymi, zaś spośród pozostałych, polegających na zastępowaniu pewnych końcówek wyrazu innymi, wykonywana jest co najwyżej jedna.

Słowo otrzymane po wykonaniu drugiego kroku stanowi wynik działania algorytmu. Czasem stosuje się wersję rekurencyjną algorytmu Lovins (zwaną też wersją iteracyjną), która polega na wielokrotnym stosowaniu algorytmu do stemowanego słowa, aż do osiągnięcia stanu, w którym kolejne wykonanie algorytmu nie powoduje już żadnych zmian.

Algorytm Lovins dzięki swej prostocie jest łatwy do implementacji i stosunkowo szybki w działaniu. Obsługuje też poprawnie niektóre niestandardowe sposoby tworzenia liczby mnogiej (np. *Unix* → *Unices*) oraz podwajanie pewnych głosek przy tworzeniu słów pochodnych (np. *fog* → *foggy*, *sit* → *sitting*). Jego wadę stanowi lista końcówek, która pomimo swoich rozmiarów nie jest kompletna i pomija pewne przyrostki słów, np. `ement`, co skutkuje czasem wygenerowaniem w prostych sytuacjach różnych rdzeni dla słów posiadających ten sam rdzeń gramatyczny.

2.5.2. Algorytm Portera

Podstawowa idea algorytmu Portera odbiega znacznie od założeń przyjętych w algorytmie Lovins. Stemming wykonywany jest nie w dwóch, lecz w aż ośmiu krokach algorytmu, przy czym dłuższe przyrostki obcinane są etapami w kolejnych krokach ([Porter]). Lista końcówek jest znacznie krótsza niż wykorzystywana w algorytmie Lovins, ale większe jest znaczenie wyrażonych bezpośrednio w kodzie instrukcji warunkowych.

O ile w algorytmie Lovins niektóre spośród 29 warunków wymuszają nieusuwanie pewnych końcówek wtedy, gdy pozostawiony rdzeń byłby krótszy od ustalonej minimalnej liczby znaków, algorytm Portera wykorzystuje w podobnym celu pojęcie miary słowa. Dla celów opisu tego algorytmu definiuje się spółgłoskę (*consonant*) jako dowolną literę inną niż `A`, `E`, `I`, `O`, `U` lub `Y` poprzedzone spółgłoską. Każdą literę nie będącą spółgłoską określa się mianem samogłoski (*vowel*). Znakiem `C` oznacza się ciąg jednej lub więcej spółgłosek, zaś znakiem `V` ciąg jednej lub więcej samogłosek. Każde słowo można zatem przedstawić w postaci `[C](VC){m}[V]`. Liczbę `m` nazywamy miarą (*measure*) tego słowa.

Kolejne kroki algorytmu przypominają nieco pierwszy krok algorytmu Lovins w tym, że pewna lista końcówek jest przeglądana w kolejności i pierwsza pasująca końcówka jest usuwana, o ile spełniony jest skojarzony z nią warunek. W odróżnieniu jednak od owego algorytmu, warunki nie dotyczą długości rdzenia wyrażonej w znakach, lecz jego miary, zaś oprócz sprawdzania, czy rdzeń kończy się pewnymi kombinacjami znaków, niektóre warunki badają występowanie w rdzeniu pewnych kombinacji spółgłosek i samogłosek. Długie końcówki są odcinane od rdzenia stopniowo, w kolejnych krokach algorytmu.

2.5.3. Inne algorytmy

Język angielski jako z jednej strony posiadający stosunkowo prostą gramatykę, a z drugiej stanowiący język o szczególnym znaczeniu w komunikacji międzynarodowej, doczekał się wielu prób stworzenia algorytmicznych metod stemmingu. Oprócz opisanych powyżej algorytmów Lovins i Portera powstało również wiele innych, m.in.: algorytm Dawsona, stanowiący rozszerzenie algorytmu Lovins, algorytm Paice-Huska, w którym podczas każdej iteracji rozpatrywana jest jedynie ostatnia litera rdzenia i algorytm Krovetza pozwalający na usunięcie końcówek związanych z tworzeniem różnych form gramatycznych danego słowa (np. tworzenie liczby mnogiej przez dodanie na końcu litery `s` lub dodanie końcówki `ing` do czasownika), używany czasem przed innymi algorytmami jako wstępny etap stemmingu.

Martin Porter, twórca algorytmu Portera, jest także autorem projektu Snowball ([Snowball]). Snowball to specjalny język programowania stworzony z myślą o implementacji algorytmów stemmingu opartych na usuwaniu przyrostków (*suffix stripper grammar*). Oprócz samej implementacji tego języka, na internetowej stronie projektu dostępne są zapisane w Snowballu algorytmy stemmingu dla kilkunastu języków europejskich oraz program pozwalający na przetłumaczenie programów w języku Snowball na ANSI C. Projekt dostępny jest na licencji typu BSD.

Egothor[1], pakiet oprogramowania umożliwiający tworzenie wyszukiwarek dla witryn internetowych, posiada wbudowany stemmer, który może zostać dopasowany do gramatyki dowolnego języka ([Bialecki]). W wyniku uczenia algorytmu poprzez podanie zbioru przykładowych słów i pasujących do nich lematów, tworzony jest zbiór reguł, które mogą potem zostać zastosowane do przekształcania również słów nie znajdujących się w zbiorze uczącym. Po jednorazowym wygenerowaniu zestawu reguł dla danego języka, może on być wykorzystywany dowolną ilość razy do stemmingu tekstów w tym języku. Podejście to można określić mianem semi-algorytmicznego, gdyż wygenerowany zestaw reguł wprawdzie trudno nazwać słownikiem, ale rozmiarem i złożonością przewyższa on zwykle znacznie zestawy reguł wykorzystywane w rozwiązaniach czysto algorytmicznych tworzonych przez ludzi. Pakiet jest dostępny na licencji w stylu Apache.

1 http://www.egothor.org/

Stosunkowo duża w porównaniu z językiem angielskim złożoność języka polskiego utrudnia tworzenie algorytmicznych rozwiązań problemu stemmingu. Dostępnych jest wiele rozwiązań opartych o metody słownikowe, w większości są to jednak produkty o zamkniętym kodzie źródłowym. W domenie publicznej znajduje się słownikowy stemmer Lametyzator/Stempelator autorstwa Dawida Weissa, oparty na zasadach polskiej fleksji zawartych w słowniku ispell i wykorzystujący automaty skończenie stanowe do przechowywania słownika ([Weiss]), jednak do swojego działania wymaga on zamkniętego pakietu FSA. W ramach projektu Stempel[2] dostępne są tablice reguł języka polskiego dla stemmera Egothor, ale ich rozszerzenie nie jest możliwe ze względu na niedostępność kodu źródłowego. Zbiory reguł wygenerowane na podstawie rozszerzonego zbioru uczącego, a więc umożliwiające zwiększenie skuteczności działania stemmera, dostępne są w wersji komercyjnej.

2.5.4. Użycie SoundexPL zamiast stemmingu dla języka polskiego

Ze względu na znaczną złożoność polskiej gramatyki, budowa prostego algorytmicznego stemmera języka polskiego jest zadaniem bardzo trudnym. W niniejszym rozdziale przedstawiona zostanie alternatywa do stemmingu, jaką jest zastosowanie opisanego w punkcie 2.5.4.2 algorytmu SoundexPL.

W proponowanym podejściu, czterocyfrowy kod SoundexPL słowa jest uznawany za jego rdzeń. Algorytm SoundexPL jest w stanie w wielu przypadkach trafnie sprowadzić do identycznego kodu różne formy gramatyczne tego samego słowa a nawet różne słowa pochodzące od tego samego rdzenia. Potrafi przy tym uwzględnić wiele występujących w języku polskim oboczności. Przykłady trafnie sprowadzonych do jednego kodu form obejmują np.: *samochód*, *samochody*, *samochodowy* (kod SoundexPL 2573); *kwiat*, *kwiaty*, *kwiatom* (kod 7130); *praca*, *pracuj* (kod 1620); *czekać*, *czekacie*, *czekasz* (kod 2720); *kolec*, *ukłuć* (kod 7420).

Główną wadą algorytmu SoundexPL jako alternatywy do stemmingu jest brak obcinania długich końcówek oraz nawet krótkich końcówek nie będących samogłoskami i odróżniających się brzmieniowo od ostatniej głoski rdzenia. Tak więc wracając do przykładów z poprzedniego paragrafu można łatwo zauważyć, że już forma *pracujesz* daje inny kod niż *pracuj*, a *czekam* inny niż *czekać*.

2 http://getopt.org/stempel/index.html

Problemem jest także nadmierne skracanie słów krótkich, zawierających wiele liter z grupy o numerze 0, co skutkuje szczególnie częstym sprowadzaniem do tego samego kodu słów niepodobnych (np. *Janek* i *noga*). Zwłaszcza litera J stanowi problem, gdyż w pewnych słowach jest ona niemal bezdźwięczna, podczas gdy w innych jest istotna dla brzmienia słowa. Zaliczenie jej do grupy 0 skutkuje np. sprowadzeniem do identycznego kodu niepodobnych słów *Jagiełło* i *kłuj*, zaś niezaliczenie jej do tej grupy powodowałoby przypisanie odmiennych kodów niektórym brzmiącym podobnie parom słów jak np. *mój* i *moi*. Można przypuszczać, że w pewnych sytuacjach wskazane może być stosowanie zmodyfikowanej wersji SoundexPL, traktującej literę J w sposób odmienny od opisanego w niniejszej pracy.

Z drugiej strony, ograniczona i ustalona z góry długość kodów Soundex i SoundexPL skutkuje przypisaniem identycznych kodów wszystkim słowom zaczynającym się od takiej samej lub podobnej brzmieniowo kombinacji liter, o ile tylko niektóre z tych słów są odpowiednio długie. W wielu przypadkach słowa takie faktycznie pochodzą od jednego rdzenia, ale często tak nie jest i przypisanie im tego samego kodu należy uważać za błąd algorytmu (np. identyczny kod 2361 mają słowa *astrofizyka* i *zdrowie*).

Całkiem odrębną klasę zastosowań może stanowić wykorzystanie kodów SoundexPL w celu sprowadzenia do znormalizowanej postaci słów zawierających błędy ortograficzne oraz słów pochodzących z dialektów, bądź w inny sposób odbiegających od normy językowej. O ile istnieje wiele algorytmów służących do badania i poprawiania błędów pisowni, o tyle zagadnienie normalizacji słów charakteryzujących się anomaliami fonetycznymi nie doczekało się jeszcze standardowych metod postępowania.

SoundexPL w większości przypadków sprowadza do tego samego kodu formę słowa zgodną z polską normą językową i różnego rodzaju odmiany fonetyczne występujące w dialektach różnych dzielnic polski (opisy dialektów za [Bartnicka]). Przykładowo, wszelkie różnice wynikające z mazurzenia, zjawiska fonetycznego polegającego na wymawianiu *sz* jak *s*, *ż* jak *z*, *cz* jak *c* i *dż* jak *dz*, są przez SoundexPL niwelowane. Podobnie jest ze znaczną częścią innych zmian wynikających ze stosowania dialektów: wielkopolskiego (ścieśnianie samogłosek i ich dyftongiczna wymowa) i śląskiego (tzw. sziakanie, wymowa samogłoski *ę* bliska *a*, zanik nosowości na końcu wyrazu) oraz

mazowieckiego (wymowa *ą* jak *o* i *ę* jak *e*). Ponieważ teksty pisane w dialektach są raczej rzadkością, zwłaszcza jeśli chodzi o dokumenty w postaci elektronicznej, ten rodzaj zastosowań algorytmu SoundexPL należy uznać raczej za ciekawostkę niż za rozwiązanie użyteczne w praktyce.

2.5.4.1. Algorytm Soundex

Algorytm Soundex został stworzony i opatentowany w USA na początku XX wieku z myślą o tworzeniu wykazów nazwisk uporządkowanych nie alfabetycznie według pisowni, lecz według brzmienia ([Russell]). Zasady angielskiej pisowni w wielu przypadkach pozwalają na zapis słów o identycznym brzmieniu na kilka różnych sposobów, co może zwłaszcza w odniesieniu do nazw własnych, nie posiadających kontekstu w innych przypadkach pozwalającego wybrać odpowiedni wariant pisowni słowa, znacznie utrudnić wyszukiwanie informacji nawet w uporządkowanych alfabetycznie wykazach.

Zastosowanie algorytmu Soundex podczas spisów ludności w Stanach Zjednoczonych wydatnie uprościło odszukiwanie danych i przyczyniło się do wzrostu jego popularności oraz wykorzystania w dziedzinach związanych z genealogią i badaniami historycznymi. Sam algorytm z biegiem lat podlegał różnorakim usprawnieniom i używana obecnie wersja różni się od jego pierwotnej postaci. Ta właśnie współczesna wersja została opisana poniżej.

Soundex przypisuje każdemu słowu pewien kod składający się z jednej litery oraz trzech cyfr. Słowa brzmiące tak samo lub podobnie powinny otrzymywać ten sam kod Soundex. Gdy algorytm jest wykorzystywany do indeksowania nazwisk, spis jest dzielony na uporządkowane leksykograficznie według kodu Soundex grupy pozycji o tym samym kodzie, zaś w ramach każdej grupy nazwiska są sortowane alfabetycznie.

Działanie algorytmu przebiega według następującego schematu:

1. Począwszy od drugiej litery słowa i posuwając się od strony lewej ku prawej, zastępuj każdą literę odpowiadającym jej kodem, zgodnie z tabelą 1.

2. Przeglądając słowo w sposób taki sam, jak w punkcie 1, usuwaj wystąpienia kodu 0.
3. Przeglądając znowu słowo w sposób opisany w punkcie 1, zastępuj wielokrotne kolejne wystąpienia każdego z kodów wystąpieniem pojedynczym.
4. Jeżeli otrzymany kod jest krótszy od 4 znaków, uzupełnij go zerami do długości 4. Jeśli kod jest dłuższy niż 4 znaki, obetnij go do długości 4 znaków.
5. Kod otrzymany w kroku 4 stanowi kod Soundex słowa otrzymanego na wejściu algorytmu.

Kod	*Litery*
0	A, E, H, I, O, U, W, Y
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Tabela 1: Kody Soundex

W opisanej postaci algorytm posiada pewne wady, którym starano się zaradzić poprzez wprowadzenie różnorakich modyfikacji. Fakt pozostawienia pierwszej litery słowa bez zmian powoduje, że pewne dwuliterowe kombinacje nie są interpretowane właściwie, jeśli występują na początku słowa – np. `PH` i `F` na początku słowa nie są uznawane za równoważne fonetycznie, chociaż są, gdy występują w dalszej części wyrazu. Aby zapobiec temu zjawisku, w wielu odmianach algorytmu Soundex albo wykonuje się krok wstępny, podczas którego niektóre dwuznaki na początku słowa są zastępowane równoważnymi im pojedynczymi literami, albo rozpoczyna się zamianę nie od drugiej, a już od pierwszej litery, co ma podobny skutek. [Prosise] proponuje by zamienić kolejnością kroki 2 i 3, aby dwie podobne spółgłoski oddzielone samogłoską nie były redukowane do pojedynczej cyfry. Istnieją również warianty, w których zmieniono długość kodu z 4 na inną liczbę lub dokonano zmian w tabeli przypisującej znaki do poszczególnych grup. Algorytm doczekał się też rozszerzenia w postaci algorytmu Soundex Daitcha-Mokotoffa ([Mokotoff]), zastosowanego po raz pierwszy przy indeksowaniu nazwisk imigrantów przybywających do Palestyny i dostosowanego do specyfiki nazwisk słowiańskich i germańskich zapisanych zgodnie z zasadami ortografii oryginalnej lub angielskiej.

Biorąc pod uwagę jego prostotę, można uznać, że wyniki zwracane przez algorytm Soundex są w większości sytuacji bardzo dobre, zaś jego wykorzystanie w praktyce potwierdziło jego przydatność. Istnieje jednak pewna liczba sytuacji, w których algorytm (zarówno oryginał jak i różne jego warianty) albo błędnie zwraca różne kody

dla słów o identycznej wymowie (np. *wood* i *would*), albo przypisuje ten sam kod słowom o wymowie znacznie się różniącej (np. *society* i *scud*).

2.5.4.2. Algorytm SoundexPL

Algorytm SoundexPL stworzony został przez autora niniejszej pracy jako proste przeniesienie zasad algorytmu Soundex na grunt języka polskiego. SoundexPL opiera się na wersji algorytmu Soundex zawierającej niektóre z opisanych wyżej modyfikacji i korzysta z dopasowanej do zasad polskiej wymowy tabeli kodów. Został także dodany krok wstępnego przetwarzania, którego celem jest poprawne uwzględnienie dwu- i trójznaków oraz zmiękczającego działania litery `i` po spółgłosce. Kody otrzymywane w algorytmie SoundexPL nie są zgodne z kodami oryginalnej ani zmodyfikowanych wersji algorytmu Soundex, tzn. słowa o identycznym brzmieniu zapisane odpowiednio według zasad pisowni angielskiej i polskiej nie otrzymują w wyniku działania tych algorytmów identycznych kodów.

Schemat działania algorytmu jest następujący:

1. Zastąp kombinacje wieloznakowe pojedynczymi literami zgodnie z tabelą 2, zachowując przedstawioną w niej kolejność zamian. Zastąpienia powinny każdorazowo obejmować cały wyraz, np. w słowie *książka* najpierw *si* jest zastępowane przez *ś*, a potem *kś* przez *x*.
2. Począwszy od pierwszej litery słowa i posuwając się od strony lewej ku prawej, zastępuj każdą literę odpowiadającym jej kodem, zgodnie z tabelą 3.

Kombinacje	***Litera***
CH	H
TRZ, TCZ, CZ, SZ	C
RZ, DZ, DŻ, DŹ	Ż
CI	Ć
NI, MI	Ń
SI	Ś
ZI	Ź
ON, OM	Ą
KS, KŚ	X

Tabela 2: Zamiana wieloznaków

3. Przeglądając znowu słowo w sposób opisany w punkcie 2, zastępuj wielokrotne kolejne wystąpienia każdego z kodów wystąpieniem pojedynczym.
4. Przeglądając słowo w sposób taki sam, jak w punkcie 2, usuwaj wystąpienia kodu 0.
5. Jeżeli otrzymany kod jest krótszy od 4 znaków, uzupełnij go zerami do długości 4. Jeśli kod jest dłuższy niż 4 znaki, obetnij go do długości 4 znaków.

6. Kod otrzymany w kroku 5 stanowi kod SoundexPL słowa otrzymanego na wejściu algorytmu.

Algorytm SoundexPL cechuje się nieco silniejszą niż oryginalny Soundex tendencją do przypisywania tego samego kodu słowom, których wymowa jest tylko odrobinę zbliżona lub zgoła całkiem niepodobna. Ta jego cecha jest jednak w zastosowaniu przedstawionym w niniejszej pracy raczej zaletą niż wadą. W pewnych sytuacjach może ona jednak stanowić utrudnienie, zwłaszcza gdy przypisane do jednego kodu słowa nie tylko pod względem fonetycznym, ale i gramatycznym mają ze sobą niewiele wspólnego (np. zarówno słowo *książka* jak i *gęsiego* mają kod SoundexPL równy 7270).

Kod	*Litery*
0	A, E, I, J, O, U, Y, Ą, Ę, Ó
1	B, F, P, V, W
2	C, S, Z, Ć, Ś, Ż, Ź
3	D, T
4	L, Ł
5	M, N, Ń
6	R
7	G, H, K, Q, X

Tabela 3: Kody SoundexPL

2.6. Wykorzystanie tezaurusa

Wykorzystanie tezaurusa, czyli słownika wyrazów bliskoznacznych, ma na celu poprawę jakości grupowania dokumentów poprzez zastąpienie wszystkich słów o podobnym znaczeniu pojedynczym słowem. Przykładowo, dwa dokumenty, z których jeden zawiera wielokrotnie słowo *kolor*, zaś w drugim często pojawia się wyraz *barwa* mogłyby nie zostać uznane za podobne, lecz po zamianie wystąpień obu tych słów na słowo *kolor*, ich podobieństwo wykrywane przez program powinno znacznie wzrosnąć.

Słownik wyrazów bliskoznacznych może również zostać użyty w przypadku, gdy pewne słowo posiada więcej niż jeden wariant pisowni, np. angielskie słowo pisane *colour* w języku angielskim brytyjskim i *color* w angielskim amerykańskim.

Sposób działania tezaurusa przypomina bardzo sposób działania lematyzatorów słownikowych, z tą różnicą, że kryterium uznania pewnych słów za równoważne jest nie pokrewieństwo gramatyczne, lecz semantyczne. O ile więc możliwe jest tworzenie lematyzatorów algorytmicznych, o tyle w przypadku słów w żaden sposób nie podobnych, lecz posiadających zbliżone znaczenie, trudno jest uniknąć korzystania ze słownika.

Wykorzystanie lematyzatora bądź stemmera algorytmicznego w połączeniu ze słownikiem wyrazów bliskoznacznych wymaga rozwagi. Istotna jest kolejność stosowania obu filtrów, gdyż w zależności od tego, czy jest stosowany przed, czy po stemmerze, tezaurus powinien jako hasła zawierać albo wyrazy w różnych formach gramatycznych, albo tylko w formie podstawowej. Dodatkowo, jeżeli stemmer produkuje rdzenie nie będące lematami, konieczne może się okazać umieszczenie w tezaurusie haseł nie będących poprawnymi słowami, a rdzeniami produkowanymi przez stemmer.

Z drugiej strony, tezaurus może stanowić prostą namiastkę lematyzatora słownikowego. Umieszczenie w nim różnych form gramatycznych często spotykanych słów, które przez stemmer nie są przetwarzane poprawnie (np. różnego rodzaju wyjątków), może doskonale uzupełnić działanie stemmera algorytmicznego.

Uwagi wymaga także kwestia umieszczania w słowniku całych zwrotów, składających się z wielu wyrazów. Różnego rodzaju związki frazeologiczne często zawierają słowa normalnie umieszczane na stop-liście, a więc jeżeli tezaurus ma je obsługiwać, powinien zostać zastosowany przed usunięciem z dokumentu tego rodzaju wyrazów. Pewne związki frazeologiczne mogą być rozdzielane na niezależne człony, występujące w zdaniu daleko od siebie (np. *iść gęsiego* → *szliśmy wąską ścieżką gęsiego*) i ich odkrycie w zdaniu może nie być proste. Zwykle tylko te wyrażenia wielowyrazowe, które nie podlegają rozdzieleniu (np. *na piechotę*), rzeczywiście poprawiają jakość działania tezaurusa na dokument.

Podobnie jak stemmery, a w odróżnieniu od lematyzatorów, słownik wyrazów bliskoznacznych (przynajmniej w podstawowej postaci) nie interpretuje w żaden sposób kontekstu, w jakim słowa występują w dokumencie. Aby działać całkowicie poprawnie, tezaurus musiałby uwzględniać nie tylko budowę gramatyczną zdania, jak lematyzator, ale również jego semantykę. Zjawisko homonimii, czyli posiadania przez jeden wyraz kilku odrębnych znaczeń, stanowi zatem największą przeszkodę w jego poprawnym działaniu.

Istnieje kilka sposobów rozwiązywania problemu homonimii. Najprostszym z nich jest nieumieszczanie w słowniku słów posiadających używane często homonimy. W wielu językach, w szczególności w języku angielskim, homonimia stanowi jednak

zjawisko bardzo powszechne i usunięcie słów posiadających homonimy z tezaurusa pozostawiłoby w nim bardzo niewiele wyrazów innych niż nazwy własne.

Inną możliwość stanowi połączenie wszystkich grup synonimów zawierających homonimy pewnego słowa w pojedynczą grupę. Skutkuje to jednak tym, że słownik wyrazów bliskoznacznych często sprowadza do tego samego słowa wyrazy posiadające zupełnie odmienne znaczenia. W językach takich jak angielski może to doprowadzić do podziału całego zbioru wyrazów zawartych w słowniku na kilka zaledwie grup synonimów – w takim przypadku działanie tezaurusa zbyt mocno zmienia dokument, aby można je było uznać za użyteczne.

Wydaje się, że optymalnym spośród wymienionych prostych rozwiązań jest unikanie umieszczania w słowniku wyrazów bliskoznacznych słów posiadających często spotykane homonimy, a jeśli takie słowa już się w słowniku znajdą – nie używanie ich jako słów zastępujących w dokumencie pozostałe synonimy z danej grupy. W przypadku, gdy dokument jest poddawany działaniu tezaurusa przed działaniem na nim lematyzatora, słownik może zawierać te formy posiadających homonimy słów, które uniemożliwiają pomylenie poszczególnych homonimów.

Słowniki wyrazów bliskoznacznych często stanowią element edytorów tekstu, a odpowiednie zbiory danych dla wielu języków są łatwo dostępne w internecie. Słowniki w takiej postaci zwykle nadają się jednak niezbyt dobrze do automatycznego stosowania, gdyż nie uwzględniają problemu homonimii, pozostawiając wybór rzeczywistych synonimów do decyzji człowieka. Aby tego rodzaju zbiory danych nadawały się do stosowania automatycznego, powinny zostać przetworzone zgodnie z uwagami podanymi w poprzednich paragrafach. W przeciwnym razie wykorzystanie tezaurusa może wręcz negatywnie wpływać na dalsze etapy przetwarzania dokumentu.

Znacznie bardziej zaawansowane podejście do stosowania tezaurusa wiąże się z przeprowadzeniem analizy gramatycznej i semantycznej zdania. Gdy tego rodzaju informacje są dostępne, możliwe staje się wyszukiwanie synonimów z uwzględnieniem znaczenia danego słowa w tekście. Przykładem zbioru danych, który może zostać w tym celu wykorzystany, jest WordNet. Stanowi on zbiór angielskich słów podzielonych na grupy synonimów z uwzględnieniem homonimii ([WordNet]). O ile możliwe jest stwierdzenie, z którym spośród znanych znaczeń danego słowa mamy do czynienia w analizowanym zdaniu, możliwe jest poprawne znalezienie synonimów tego słowa oraz

innych semantycznie z nim związanych słów (np. innych części mowy), bez niebezpieczeństwa błędnej interpretacji homonimów. W praktyce znacznie częściej niż skomplikowane algorytmy tezaurusa stosowane są metody wykrywania homonimii takie jak LSA, oparte na znajdowaniu słów występujących często wspólnie i odgadywaniu na tej podstawie ich znaczeń.

2.7. Ważenie wyrazów

Poszczególnym wyrazom wchodzącym w skład dokumentu przypisuje się często wagi, określające jak bardzo każde poszczególne wystąpienie pewnego słowa jest istotne dla treści całego tekstu. Nie wszystkie słowa są równie ważne dla określenia jego tematyki. Słowa należące do stop-listy danego języka w ogóle nie wpływają na treść dokumentu (patrz punkt 2.4), zaś wyrazy występujące np. w tytułach zwykle uznaje się za istotne nawet wtedy, gdy nie występują zbyt często w innych miejscach.

Wagi mogą być obliczane na podstawie różnych kryteriów, z których kilka przedstawiono w następujących podpunktach. W przypadku, gdy stosowanych jest równocześnie kilka metod obliczania wag, końcowa waga może stanowić iloczyn albo sumę wag częściowych obliczonych za pomocą różnych algorytmów.

Po przypisaniu słowom wag, wagi te mogą zostać wykorzystane w rozmaity sposób przez algorytmy przetwarzające dokument. W przypadku algorytmów traktujących dokument jako *bag of words*, wartości odpowiednich liczników mogą być zwiększane o wagę każdego napotkanego słowa, zamiast o jeden. W przypadku algorytmów zliczających n-gramy (patrz punkt 3.2.2), można na podstawie wag słów modyfikować liczniki n-gramów.

2.7.1. Znaczniki HTML

HTML jest najpopularniejszym formatem dokumentów tekstowych dostępnych w sieci WWW. Oprócz samego tekstu, dokumenty HTML zawierają również znaczniki, które dostarczają dodatkowych informacji o dokumencie.

Znaczniki zawarte w ciele dokumentu wyznaczają sposób formatowania poszczególnych fragmentów tekstu oraz rolę, jaką te fragmenty pełnią. Wraz ze wzrostem popularności arkuszy stylów, w ostatnich latach zmienił się sposób tworzenia stron WWW i upowszechniło się tworzenie tych stron w taki sposób, by oddzielić w znacznej mierze informacje o wyglądzie tekstu od jego treści. Wiąże się to z wykorzysta-

niem arkuszy stylów do formatowania i używaniem w dokumentach tylko ograniczonego zestawu znaczników, z których większość kojarzona jest z pewnymi logicznie określonymi częściami dokumentu (np. H1-H6 dla nagłówków). Tak wyróżnione części tekstu zasługują na szczególną uwagę przy analizie dokumentu.

Prostym przykładem wykorzystania znaczników HTML do przypisania wag słowom dokumentu może być skojarzenie poszczególnych znaczników z wagami, a następnie przypisywanie każdemu wyrazowi wagi znacznika, w którego wnętrzu wyraz ten się znajduje. W przypadku zagnieżdżonych znaczników wagi te mogą kumulować się, np. poprzez dodawanie lub mnożenie. Zwykle przypisuje się wyższe niż przeciętna wagi znacznikom takim jak H1-H6, które służą do zaznaczania logicznego podziału dokumentu na części, lub STRONG, służącemu do wyróżniania w tekście szczególnie istotnych słów. Często uwzględnia się także znaczniki służące jedynie do graficznego wyróżnienia fragmentów tekstu, np. B oraz I.

Dokumenty HTML posiadają nagłówek, w którego skład wchodzą m.in. znaczniki: TITLE (tytuł dokumentu) oraz META (dodatkowe informacje o dokumencie takie jak: kodowanie znaków, słowa kluczowe i opis). Informacjom z nagłówka, zwłaszcza tytułowi, przypisuje się zwykle duże znaczenie przy klasyfikacji dokumentu. Tytuł, słowa kluczowe i opis mogą w zależności od przyjętego modelu dokumentu albo być rozpatrywane jako metadane, oddzielone od samej treści dokumentu, albo zostać w wewnętrznej reprezentacji dokumentu dołączone do pozostałego tekstu (z odpowiednio wysokimi wagami).

2.7.2. Informacje o gramatyce

Do określenia tematyki dokumentu najbardziej przyczyniają się rzeczowniki i czasowniki, podczas gdy np. spójniki są w większości przypadków zupełnie nieistotne i często znajdują się na stop-listach. Stąd jednym z zastosowań informacji o budowie gramatycznej zdań wchodzących w skład dokumentu może być zmiana wag słów. Możliwych jest wiele wariantów tej metody, poczynając od najprostszej, polegającej na prostym przypisaniu wag poszczególnym częściom mowy, poprzez wykorzystanie również informacji o budowie gramatycznej całego zdania (np. celem zwiększenia wagi podmiotu i orzeczenia) aż do skomplikowanych algorytmów analizujących wiele różnych aspektów zdania (np. szyku jako sposobu podkreślenia wybranych części wypowiedzi).

2.7.3. Inne własności tekstu

Czasem jako wyznacznik wagi słowa stosuje się jego pozycję w dokumencie. W takim przypadku zwykle największą wagę przypisuje się słowom znajdującym się na początku dokumentu, a słowom znajdującym się bliżej końca – wagę mniejszą. Uzasadnieniem takiego postępowania jest założenie, że na początku dokumentu znajdują się zwykle tytuły oraz streszczenia, zawierające wiele słów charakterystycznych dla dokumentu. Założenie takie nie zawsze jest prawdziwe, a bardziej elastyczne metody wyznaczania ważnych fragmentów tekstu, np. za pomocą znaczników HTML, generalnie oferują większą szansę poprawnego przypisania wag. Z drugiej strony, ponieważ metoda przypisywania początkowi dokumentu większej wagi niż jego końcowi była i jest nadal stosowana przez wiele wyszukiwarek internetowych, znaczna część stron WWW została dostosowana do tego wymagania i rzeczywiście zawiera nagromadzenie charakterystycznych wyrazów na początku dokumentu.

Do określania wag słów używa się też czasem informacji o sposobie ich zapisu. Zwiększoną wagę mogą otrzymać słowa pisane wielkimi literami lub słowa wyróżnione sposobami używanymi na listach dyskusyjnych, gdzie umieszczenie słowa między dwoma gwiazdkami używane jest w miejsce *pogrubienia*, a otoczenie go znakami podkreślenia oznacza _podkreślenie_.

3. Generacja współrzędnych

Wstępne przetwarzanie tekstów umożliwia usunięcie z nich znacznej ilości zbędnych informacji, ale dokumenty reprezentowane przez ciągi zdań i słów mimo to trudno jest między sobą porównywać. Aby dokumenty tekstowe mogły zostać poddane procesowi grupowania lub klasyfikacji, wygodnie jest przedstawić je w postaci pewnych obiektów, których porównywanie ze sobą jest proste i szybkie. W modelu VSM stosuje się tym celu reprezentację dokumentów w postaci wektorów liczb rzeczywistych. Wektory reprezentujące dokumenty mogą już stosunkowo prosto zostać zgrupowane z użyciem jednej z powszechnie stosowanych metod.

3.1. Przestrzenie metryczne

Najczęściej celem reprezentacji dokumentów w przestrzeniach wektorowych jest porównywanie ich między sobą oraz ich grupowanie. Metryki pozwalają na obliczanie wykorzystywanych w celu porównywania dokumentów odległości między odpowiadającymi im wektorami.

Pojęcie przestrzeni metrycznej nie jest związane w żaden sposób z pojęciem przestrzeni wektorowej, jednak w VSM niemal zawsze mamy do czynienia ze zbiorami punktów, które stanowią zarówno przestrzeń wektorową jak i metryczną. Niektóre przedstawione poniżej metryki (np. metrykę Hamminga) można uogólnić tak, by można je było stosować w przestrzeniach, których punkty nie stanowią ciągów liczb, lecz dowolnych symboli (np. znaków). Znajdują one zastosowanie przy porównywaniu nie tylko całych dokumentów (reprezentowanych przez wektory), lecz także pojedynczych słów i są czasem wykorzystywane w algorytmach wstępnego przetwarzania tekstów. Przykładowo, metryka Levenshteina jest wykorzystywana w pewnych algorytmach poprawy błędów ortograficznych oraz w stemmerze Egothor podczas procesu uczenia ([Bialecki]).

3.1.1. Definicja

Przestrzenią metryczną (X,d) nazywamy zbiór X wyposażony w funkcję $d: X \times X \to \mathbb{R}$ spełniającą dla dowolnych $x,y,z \in X$ warunki:

- $d(x,y) \geqslant 0$
- $d(x,y)=0 \Leftrightarrow x=y$

- $d(x,y)=d(y,x)$
- $d(x,z)\leqslant d(x,y)+d(y,z)$ (nierówność trójkąta)

Elementy zbioru X nazywamy punktami, zaś funkcję d metryką przestrzeni. Metryka interpretowana jest jako rozszerzenie intuicyjnego pojęcia odległości między punktami.

3.1.2. Przegląd metryk

Dokumenty tekstowe najczęściej reprezentowane są jako wektory w przestrzeniach liniowych $\mathbb{R}^n$, zatem właśnie metryki zdefiniowane w tego rodzaju przestrzeniach są z punktu widzenia niniejszej pracy najbardziej interesujące. W podanych wzorach n oznacza wymiar przestrzeni, zaś $\mathbf{x}=(x_1,\ldots,x_n)$ i $\mathbf{y}=(y_1,\ldots,y_n)$ są należącymi do niej wektorami. Poniżej przedstawiono niektóre metryki stosowane w zagadnieniach związanych z przetwarzaniem dokumentów tekstowych.

- Metryka euklidesowa, zgodna z potocznym rozumieniem słowa „odległość", określona jest wzorem
  $$d(\mathbf{x},\mathbf{y})=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$
  Ponieważ obliczanie pierwiastka kwadratowego jest operacją stosunkowo powolną, w praktyce często stosuje się różnego rodzaju przybliżone metody obliczania wartości tej funkcji.
- Metryka Mahattan:
  $$d(\mathbf{x},\mathbf{y})=\sum_{i=1}^{n}|x_i-y_i|$$
- Metryka maksimum:
  $$d(\mathbf{x},\mathbf{y})=\max_{i=1,\ldots,n}|x_i-y_i|$$
- Metryka Minkowskiego stanowi jednoparametrową rodzinę metryk określonych wzorem
  $$d_p(\mathbf{x},\mathbf{y})=\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{\frac{1}{p}}$$
  Metryki: Manhattan i euklidesowa stanowią szczególne przypadki metryki Minkowskiego, z parametrem p równym odpowiednio 1 i 2. Metryka maksimum stanowi przypadek graniczny dla $p\to\infty$.

- Metryka Hamminga zdefiniowana jest jako
  $d(\mathbf{x},\mathbf{y})=\sum_{i=1}^{n}(1-\delta_{x_i y_i})$, gdzie δ oznacza deltę Kroneckera: $\delta_{ij}=\begin{cases}1, & \text{gdy } i=j\\ 0, & \text{gdy } i\neq j\end{cases}$

  Metryka ta najczęściej stosowana jest w przestrzeniach, w których współrzędne wektorów należą do dyskretnego zbioru wartości. W przypadku, gdy współrzędne wektorów przyjmują wartości należące do zbioru nieprzeliczalnego, w szczególności gdy są liczbami rzeczywistymi, metryka ta rzadko znajduje zastosowanie, gdyż prawdopodobieństwo przyjęcia przez pewną współrzędną dwóch wektorów dokładnie tej samej wartości jest bliskie zeru. Można jednak wprowadzić warianty metryki Hamminga nadające się do praktycznego zastosowania dla tego rodzaju danych. Jednym z rozwiązań może być kwantyzacja wartości przyjmowanych przez współrzędne a następnie zastosowanie do nich zwykłej metryki Hamminga. Sposób przeprowadzenia kwantyzacji powinien być dopasowany do oczekiwanego rozkładu wartości poszczególnych współrzędnych wektorów. Przy pewnych rodzajach danych użyteczna może się okazać odmiana metryki Hamminga, w której „kwantyzację" przeprowadzamy w taki sposób, że wartość 0 pozostawiamy bez zmian, zaś każdą inną wartość zastępujemy przez 1.

- Metryka Levenshteina (zwana także metryką redakcyjną) najczęściej stosowana jest do łańcuchów, choć można oczywiście stosować ją również do wektorów liczb, traktując wektory jak łańcuchy, a liczby jak znaki. Odległość Levenshteina między dwoma łańcuchami jest zdefiniowana jako minimalna liczba operacji potrzebnych do zamiany jednego łańcucha na drugi, przy czym pojedynczą operacją może być wstawienie znaku, usunięcie znaku oraz zamiana znaku na inny. Istnieją warianty przypisujące różne wagi poszczególnym operacjom, bądź uwzględniające dodatkowe operacje, np. przestawienie liter.

- Metryka kosinusowa zdefiniowana jest wzorem:
  $d(\mathbf{x},\mathbf{y})=1-\frac{\mathbf{x}\cdot\mathbf{y}}{|\mathbf{x}||\mathbf{y}|}=1-\frac{\sum_{i=1}^{n}x_i y_i}{\sqrt{\sum_{i=1}^{n}x_i^2}\sqrt{\sum_{i=1}^{n}y_i^2}}$

  i jest wykorzystywana szczególnie często. Związana jest ona z kosinusem kąta między wektorami $\mathbf{x}$ i $\mathbf{y}$, określonym wzorem $\cos(\sphericalangle(\boldsymbol{x},\boldsymbol{y}))=\frac{\boldsymbol{x}\cdot\boldsymbol{y}}{|\boldsymbol{x}||\boldsymbol{y}|}$ i stanowiącym miarę podobieństwa dokumentów przez te wektory reprezentowanych.

W praktyce czasem wykorzystuje się również miary odległości nie będące metrykami (np. nie spełniające nierówności trójkąta), ale dobrze oddające podobieństwa i

różnice dokumentów reprezentowanych przez wektory. „Metryki" takie można tworzyć na przykład w oparciu o różne miary podobieństwa wykorzystywane w metodach znajdowania dokumentów pasujących do podanych słów kluczowych.

- Metryka oparta na współczynniku Jacquarda określona jest dla wektorów o współrzędnych nieujemnych przez wyrażenie:
  $$d(\mathbf{x},\mathbf{y}) = 1 - \frac{\mathbf{x}\cdot\mathbf{y}}{|\mathbf{x}|^2 + |\mathbf{y}|^2 - \mathbf{x}\cdot\mathbf{y}} = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2 - \sum_{i=1}^{n} x_i y_i}$$
  Współczynnik Jacquarda $\frac{\mathbf{x}\cdot\mathbf{y}}{|\mathbf{x}|^2 + |\mathbf{y}|^2 - \mathbf{x}\cdot\mathbf{y}}$ stanowi miarę podobieństwa dokumentów reprezentowanych przez wektory $\mathbf{x}$ i $\mathbf{y}$ ([Salton]).

3.2. Wybór bazy przestrzeni cech

Przestrzenie cech, w których reprezentowane są dokumenty tekstowe w modelu VSM, są przestrzeniami liniowymi, a zatem można mówić o bazach w tych przestrzeniach. Nazwa *przestrzeń cech* pochodzi stąd, że zwykle jako wektory bazowe wybierane są wektory zwane *wektorami cech*, związane z pewnymi elementami składowymi dokumentów (*cechami*), np. z poszczególnymi słowami lub n-gramami. Wybór ten jest w dużej mierze arbitralny i w związku z tym trudno jest interpretować w takich przestrzeniach pojęcia takie jak liniowa niezależność czy też ortogonalność. Oczywiście, pojęcia te można wprowadzić formalnie tak samo jak w „zwykłych" przestrzeniach $\mathbb{R}^n$, ale ich interpretacja w odniesieniu do dokumentów nie jest całkiem jasna. Co na przykład dokładnie miałaby oznaczać ortogonalność dwóch dokumentów? W większości przypadków, a zwłaszcza jeśli do porównywania dokumentów korzystamy z metryki kosinusowej, logiczną interpretacją wydaje się taka, że dwóm dokumentom powinny odpowiadać wektory ortogonalne, jeżeli dokumenty „dotyczą całkowicie różnych tematów". Nie jest to jednak definicja ścisła i stwierdzenie, czy pewne dokumenty dotyczą tego samego tematu, czy nie, nawet dla człowieka może nie być zadaniem ani łatwym, ani posiadającym tylko jedno poprawne rozwiązanie. Również to, co rozumiemy pod pojęciem „poprawności" rozwiązania, w dużej mierze zależy od konkretnego zastosowania technik analizy dokumentów.

Zwykle wybór bazy jest przeprowadzany na podstawie pewnej heurystyki związanej z interpretacją wektorów jako dokumentów, ale same przekształcenia wektorów wykonywane są w sposób formalny, bez wnikania w szczegóły interpretacji tych przekształceń. Często przyjmuje się przy tym dla wygody pewne założenia, które nie-

koniecznie muszą być prawdziwe. Przykładowo, wystąpienia niektórych słów zwykle bywają ze sobą skorelowane, w związku z czym odpowiadające im wektory są, w pewnym sensie, liniowo zależne, a pomimo to zbiór wektorów zawierający takie elementy często traktujemy jak bazę.

3.2.1. Baza słów

Najprostszą i często stosowaną reprezentację dokumentów stanowią wektory w przestrzeni, której wektory bazowe są związane z pewnymi słowami. Wartościami współrzędnych dla konkretnego dokumentu mogą być na przykład liczby wystąpień słów w dokumencie lub inne miary istotności poszczególnych słów dla jego treści.

Rozwiązanie takie jest proste w implementacji, ale posiada też wady. Nawet po odfiltrowaniu z tekstu słów ze stop-listy, pewne wyrazy mają tendencję do występowania w wielu znacznie się między sobą różniących dokumentach. Obecność wektorów odpowiadających takim słowom w bazie przestrzeni może zatem prowadzić w pewnych sytuacjach do zwiększenia odległości między podobnymi dokumentami, a w innych do zmniejszenia odległości między dokumentami niepodobnymi. Nasilenie tych trudności zależy w dużym stopniu od wyboru sposobu ważenia cech (patrz punkt 3.3).

Inny problem stanowi fakt wspólnego występowania niektórych wyrazów w charakterystycznych związkach frazeologicznych. Rozpatrywanie takich słów z osobna może prowadzić do wykrycia fałszywych podobieństw między różniącymi się znacznie dokumentami, gdyż ich znaczenie może być różne w zależności od kontekstu. Dość dobrym sposobem uniknięcia tego zjawiska jest wykorzystanie jako bazy zbioru wektorów skojarzonych nie z pojedynczymi słowami, a z n-gramami.

3.2.2. Baza n-gramów

Nazwą n-gram określamy ciąg (lub zbiór z powtórzeniami) n występujących w dokumencie kolejno po sobie wyrazów. W pierwszym przypadku mówimy o n-gramach uporządkowanych, w drugim – o nieuporządkowanych. Przewaga n-gramów nad pojedynczymi słowami, czyli unigramami, polega na tym, że tworzą one pewien kontekst dla słów wchodzących w ich skład. Dzięki temu możliwe jest np. wykrycie różnego rodzaju związków frazeologicznych i traktowanie ich jako całości zamiast rozbijania ich na poszczególne słowa, rozpatrywane oddzielnie.

Decyzja o stosowaniu n-gramów uporządkowanych lub nieuporządkowanych może zależeć od gramatyki języka dokumentu oraz w pewnym stopniu od stylu tekstu. N-gramy nieuporządkowane można uznać za bardziej elastyczne, gdyż za ten sam n-gram nieuporządkowany uchodzić mogą różne wyrażenia zawierające te same słowa, ale w innej kolejności. W praktyce nawet w językach o swobodnym szyku, takich jak polski, użycie n-gramów uporządkowanych i nieuporządkowanych zwykle prowadzi do zbliżonych wyników.

Dobór odpowiedniej długości n-gramów ma kluczowe znaczenie dla poprawnej reprezentacji dokumentów w przestrzeni liniowej. Unigramy, czyli pojedyncze słowa, posiadają jako wektory bazowe liczne wady wymienione w poprzednim podpunkcie. Z kolei zbyt duża wartość n prowadzi do tego, że n-gramy niemal nie powtarzają się w tekście, a zatem ich użyteczność dla rozróżniania dokumentów jest znikoma. Najczęściej używane są bi- i trigramy, czyli n-gramy o n równym odpowiednio 2 i 3. Taka długość nie powoduje jeszcze omówionych wyżej problemów związanych ze zbyt dużymi wartościami n, a równocześnie zapewnia znacznie więcej kontekstu dla słów niż unigramy.

Typowe długości n-gramów: 2 i 3 odpowiadają liczbie wyrazów wchodzących w skład większości związków frazeologicznych. Dwuwyrazowymi związkami frazeologicznymi są często pary przymiotnik-rzeczownik (*przelotne opady*, *drukarka laserowa*, *quantum mechanics*, *reverse engineering*). Trójwyrazowe związki frazeologiczne obejmują wiele nazw własnych organizacji i instytucji (np. *Narodowy Bank Polski*, *International Monetary Fund*) oraz inne często stosowane kombinacje rzeczownika i dwóch przymiotników (*dyskretna transformata kosinusowa*, *computational linear algebra*) bądź przymiotnika i dwóch rzeczowników (*obiektowy język programowania*, *public-key cryptography*).

Czasem stosowanie bi- lub trigramów może prowadzić do pominięcia słów istotnych dla rozróżniania dokumentów. Sytuacja taka zachodzi wtedy, gdy pewne słowo jest charakterystyczne dla dokumentu, ale występuje w bardzo wielu różnych związkach frazeologicznych. Wówczas pewna grupa dokumentów może zawierać to słowo, ale za każdym razem w innym n-gramie, w związku z czym podobieństwo takich dokumentów nie jest wykrywane. Dla długich dokumentów prawdopodobieństwo zajścia takiego zdarzenia jest raczej niewielkie. Poważne trudności może natomiast wywołać

korzystanie z bi- lub trigramów przy analizie zbiorów dokumentów krótkich. W takiej sytuacji może dojść do bardzo rzadkiego powtarzania się n-gramów i w efekcie do niewykrycia większości podobieństw między dokumentami.

Sposób zliczania wystąpień poszczególnych n-gramów w dokumencie powinien być zgodny z założeniem, że mają one reprezentować ciągi bądź zbiory wyrazów występujących w dokumencie wspólnie, a więc w jakiś sposób związanych ze sobą logicznie. Oznacza to między innymi, że przy traktowaniu n-gramu jako n-wyrazowego okna przesuwającego się wzdłuż tekstu, nie należy uwzględniać n-gramów składających się z ciągu słów, z których niektóre stanowią koniec jednego zdania, a pozostałe początek drugiego. Bardziej wyrafinowane metody mogą uwzględniać również wewnątrzzdaniowe znaki interpunkcyjne i np. traktować tekst zawarty w nawiasach jako wtrącenie, w którym n-gramy są zliczane oddzielnie od reszty zdania.

3.2.3. Inne możliwości wyboru bazy

Pomimo, że bazy oparte o słowa lub n-gramy należą do najczęściej stosowanych, możliwe jest również wykorzystanie w tej roli innych elementów tekstu. Przykładowo, system porównujący dokumenty pod kątem ich stylu, a nie tematu, mógłby korzystać z bazy o wektorach odpowiadających różnym częściom mowy.

Wybór bazy przestrzeni cech odpowiadającej słowom lub n-gramom jest intuicyjny dla człowieka. Możliwe jest jednak wykonanie pewnych formalnych przekształceń na wektorach w celu przejścia do innej bazy, której wektory stanowią kombinacje liniowe wektorów bazy początkowej. Pomimo, że znaczenie współrzędnych w takiej przestrzeni jest trudniejsze do zrozumienia, transformacje tego rodzaju stosuje się czasem w praktyce. Przykładem może być redukcja wymiaru przestrzeni cech z użyciem techniki PCA opisana w punkcie 3.4.2.

3.3. Ważenie cech w dokumentach

Po wyznaczeniu bazy przestrzeni cech możliwe jest wygenerowanie wektorów odpowiadających poszczególnym dokumentom. Ponieważ baza jest już ustalona, jedynym problemem do rozwiązania pozostaje znalezienie właściwej wartości dla każdej współrzędnej wektora reprezentującego konkretny tekst, czyli wagi odpowiedniej cechy dla tego dokumentu. Dla poprawnego działania algorytmów operujących na wektorach dokumentów, np. grupowania, ważne jest, aby wartości odpowiednich współ-

rzędnych były podobne dla podobnych (w sensie tematyki) dokumentów, a różniły się dla dokumentów o odmiennej treści.

3.3.1. Obecność

Najprostsza metoda przypisania wartości poszczególnym współrzędnym wektorów reprezentujących dokumenty polega na sprawdzaniu, czy skojarzone ze współrzędną słowo lub n-gram występuje w dokumencie, czy nie, i nadaniu jej odpowiednio wartości 1 lub 0. W przypadku stosowania takich wektorów o współrzędnych będących zmiennymi logicznymi, mówimy o boolowskim modelu wyszukiwania informacji. Niewątpliwą zaletą tego podejścia jest jego prostota oraz fakt, że niektóre metryki mogą dla wektorów zerojedynkowych być obliczane w uproszczony, a więc i bardziej wydajny sposób.

Wada binarnego podejścia do generacji wartości współrzędnych polega przede wszystkim na nadmiernym uproszczeniu modelu dokumentu. Fakt obecności pewnego słowa w tekście sam w sobie nie implikuje, aby tekst ten był związany tematycznie z owym słowem. Model binarny przypisuje identyczną wagę pojedynczemu wystąpieniu pewnego słowa w długim dokumencie i wielokrotnemu wystąpieniu tego samego słowa w dokumencie krótkim. Przypadkowe wystąpienia pewnych słów, np. w związkach frazeologicznych lub w nie związanych z treścią dokumentu odnośnikach na stronie WWW mogą w tym modelu bardzo silnie wpływać na postać generowanych wektorów. Problemom tym można częściowo zaradzić przypisując wartość 1 tylko tym współrzędnym, których cecha występuje w dokumencie przynajmniej pewną ustaloną, większą od jedności, liczbę razy.

3.3.2. Częstość

Częstością słowa lub innego elementu tekstu nazywamy liczbę jego wystąpień w tym tekście. Częstość względna to stosunek częstości do łącznej liczby obiektów danego rodzaju w dokumencie. Można również rozpatrywać częstość liczoną względem liczby różnych obiektów danego rodzaju pojawiających się w dokumencie, a nie wszystkich obiektów tego rodzaju w nim występujących. Użycie częstości cech jako wartości odpowiadających im współrzędnych wektora reprezentującego dokument stanowi jedno z intuicyjnie najbardziej oczywistych rozwiązań.

Obliczenie częstości cech w dokumencie wymaga przejrzenia jedynie tego jednego dokumentu. Nie są wykorzystywane żadne informacje dotyczące innych dokumentów ani nawet globalne statystyki. Dzięki temu możliwe jest przetwarzanie dowolnej liczby tekstów w ograniczonej ilości pamięci operacyjnej, o ile tylko każdy dokument z osobna może się w niej pomieścić. Prostota oraz znaczna szybkość takiego sposobu generacji współrzędnych stanowią jego niewątpliwą zaletę.

Użycie częstości bezwzględnych powoduje, że dokumenty dłuższe są reprezentowane przez zupełnie inne wektory niż dokumenty krótsze dotyczące tego samego tematu. Zaradzić temu może normalizacja wektorów albo korzystanie z metryki kosinusowej, która jest czuła jedynie na proporcje między poszczególnymi współrzędnymi, a nie na ich bezwzględne wartości. Zamiast przeprowadzania normalizacji wektorów można jako wartości współrzędnych wykorzystać częstości względne. Tak otrzymane wektory nie są znormalizowane i przez to nie leżą wszystkie na powierzchni jednej sfery. Może to pozwolić klasteryzatorowi na łatwiejsze znalezienie grup dokumentów, o ile stosowana metryka uwzględnia odległości punktów od środka układu współrzędnych.

Główną wadę stosowania częstości jako wartości współrzędnych stanowi fakt, że znaczna liczba wystąpień słowa lub n-gramu w dokumencie nie zawsze oznacza, że to słowo lub n-gram rzeczywiście jest istotne dla jego treści. Pomimo wstępnego przetwarzania dokumentów, w ramach którego usuwane są słowa ze stop-listy, wiele tekstów zawiera liczne wystąpienia słów mało istotnych dla porównywania treści różnych dokumentów między sobą. W skrajnym przypadku może się zdarzyć, że słowo występuje we wszystkich dokumentach rozpatrywanego zbioru. W takiej sytuacji co prawda w różnych dokumentach może zostać mu przypisana różna waga, ale jest wątpliwym, czy zliczanie wystąpień takiego słowa rzeczywiście pozwala na ocenę jego znaczenia dla treści poszczególnych tekstów. Jest dość prawdopodobne, że słowo to pełni jedynie rolę pomocniczą, nie związaną z tematyką dokumentów, a różne liczby jego wystąpień w poszczególnych tekstach wynikają tylko z przypadkowych odchyleń statystycznych. Wówczas wartość odpowiadającej temu słowu współrzędnej nie tylko nie ułatwia porównywania dokumentów, ale wręcz je utrudnia, gdyż wprowadza niemal losowe zaburzenia do wektorów i może doprowadzić do oddalenia się od siebie wektorów dokumentów podobnych lub do przypadkowego zbliżenia wektorów odpowiadających zupełnie różnym dokumentom.

3.3.3. TFIDF

Współczynnik TFIDF zdefiniowany jest jako iloczyn: $TFIDF = TF \cdot IDF$, gdzie TF oznacza *text frequency*, zaś IDF – *inverse document frequency*. Współczynniki te dla cechy x zdefiniowane są następująco:

- $TF = \frac{N_x}{N}$
- $IDF = \log\left(\frac{P}{P_x}\right)$,

przy czym N_x oznacza liczbę wystąpień cechy x w dokumencie, N łączną liczbę cech odpowiedniego rodzaju w dokumencie, P_x liczbę dokumentów zawierających x, zaś P – liczbę wszystkich dokumentów w zbiorze.

Głównym założeniem TFIDF jest obserwacja, że wpływ na to, jak dane słowo (lub inna cecha) jest istotne dla odróżnienia pewnego dokumentu od innych, ma nie tylko częstość słowa w tym dokumencie, ale także to, czy występuje ono w innych dokumentach, czy nie. Słowo występujące we wszystkich dokumentach pozwala tylko w bardzo niewielkim stopniu na wykrycie różnic między nimi. Słowu takiemu odpowiada współczynnik TFIDF równy 0, co jest wynikiem lepiej oddającym jego rzeczywistą wagę, niż użycie w tym miejscu samej tylko częstości słowa w dokumencie.

Ponieważ obniżanie do zera wag cech występujących we wszystkich dokumentach nie zawsze jest pożądane, często stosuje się człon IDF o postaci $IDF = 1 + \log\left(\frac{P}{P_x}\right)$. Taką postać IDF przedstawia m.in. [Salton]. Spotyka się również warianty współczynnika TFIDF korzystające ze zmienionych definicji TF, na przykład: $TF = \frac{N_x}{N_{max}}$ lub $TF = \frac{1}{2} + \frac{1}{2}\frac{N_x}{N_{max}}$ (N_{max} oznacza liczbę wystąpień w dokumencie najczęstszej w nim cechy danego rodzaju). Można także w oryginalnym wzorze w miejscu N użyć liczby różnych cech w dokumencie zamiast liczby wszystkich cech. Bez względu na różnice, wszystkie warianty TFIDF łączy jednak to, że wysoki współczynnik TFIDF otrzymuje się dla słów bądź n-gramów występujących licznie w dokumencie, ale nie występujących zbyt często w innych dokumentach. Takie właśnie cechy najlepiej pozwalają na wykrycie różnic i podobieństw między tekstami.

TFIDF jest bardzo popularnym sposobem generacji współrzędnych, gdyż łączy względną prostotę ze znacznie lepszą jakością wyników niż ważenie cech za pomocą częstości. Pewną wadę stanowi fakt, że algorytm ten korzysta w obliczeniach z war-

tości łącznej liczby dokumentów zawierających wybraną cechę, a więc czyni koniecznym przechowywanie liczników dla wszystkich cech występujących w którymkolwiek z dokumentów. W przypadku dużych zbiorów danych, liczba różnych słów (a tym bardziej n-gramów), a co za tym idzie również zajętość pamięci, może być zatem znaczna.

3.3.4. Normalizacja wektorów

W pewnych sytuacjach wektory reprezentujące dokumenty poddaje się normalizacji, czyli przekształceniu sprowadzającemu ich długość (w sensie metryki euklidesowej) do wartości 1 przy równoczesnym zachowaniu kierunku. Zabieg ten wykonywany jest głównie w przypadku stosowania metryki kosinusowej, gdyż nie zmienia on odległości między wektorami w tej metryce, natomiast pozwala zmniejszyć ilość obliczeń (wystarczy obliczać iloczyny skalarne, gdyż długości z założenia wynoszą 1).

W przypadku innych metryk, zwłaszcza metryk Minkowskiego, normalizacja wektorów bardzo znacząco wpływa na wzajemne odległości między nimi. Jej stosowanie może jednak być uzasadnione, jeśli wartości współrzędnych wektorów nie są wyskalowane w stosunku do innych (np. reprezentują bezwzględne liczby wystąpień słów w dokumencie). W takim przypadku normalizacja może mieć pozytywny skutek, gdyż sprowadza współrzędne wektorów do wartości względnych, których porównywanie jest bardziej miarodajne.

3.4. Redukcja wymiaru przestrzeni cech

Nawet dla stosunkowo niewielkich zbiorów dokumentów, liczba różnych występujących w nich n-gramów często osiąga wysokie wartości, rzędu dziesiątek tysięcy, a nawet więcej. Generowanie i późniejsze przetwarzanie tak długich wektorów często okazuje się kłopotliwe i powolne. Z tego powodu stosowane są różne metody redukcji wymiaru przestrzeni cech, których celem jest reprezentacja dokumentów tekstowych w przestrzeniach o mniejszym wymiarze.

3.4.1. Uporządkowanie według częstości

Prostą metodę zmniejszenia wymiaru przestrzeni cech stanowi uporządkowanie wektorów bazy według globalnej częstości odpowiadających wektorom bazowym obiektów, a następnie pozostawienie w bazie jedynie pewnej liczby początkowych wektorów. Jako kryterium porządkowania można także przyjąć maksimum względnych częstości cechy we wszystkich dokumentach lub sumę jej częstości względnych

we wszystkich tekstach. Opisane metody pozwalają na odrzucenie współrzędnych związanych ze słowami bądź n-gramami prawdopodobnie mało przydatnymi do rozróżniania dokumentów między sobą.

Do wad tego podejścia należą trudności omówione w punkcie 3.3.2: z jednej strony wysoka częstość słowa wcale nie musi oznaczać, że może ono być skutecznie użyte do rozróżniania dokumentów, a z drugiej niektóre słowa o niskiej częstości globalnej mogą tę cechę posiadać (na przykład gdy wszystkie wystąpienia znajdują się w jednym tylko dokumencie). Ponadto, wymaga ono znalezienia i przechowywania w pamięci przez czas działania algorytmu globalnych częstości słów, a więc neutralizuje „lokalność" stanowiącą jedną z głównych zalet wykorzystywania częstości słów jako współrzędnych.

3.4.2. PCA

Principal Component Analysis (PCA) jest techniką pozwalającą na redukcję wymiaru przestrzeni zawierającej pewien zbiór danych, w taki sposób, by zachować jak największy rozrzut między punktami w zredukowanej przestrzeni ([Mirkin]). Podstawowa transformacja danych wykonywana w ramach PCA stanowi odpowiednio dobrane przekształcenie liniowe, które pozwala na późniejsze rzutowanie punktów na przestrzeń o mniejszym wymiarze przy równoczesnym zachowaniu jak największej ilości informacji.

Aby zachować jak największą ilość informacji i umożliwić później łatwe odróżnienie poszczególnych punktów, jako pierwsza oś nowego układu współrzędnych wybierany jest w metodzie PCA kierunek, wzdłuż którego występuje największa wariancja danych. Kierunek ten nosi nazwę pierwszej składowej głównej (*principal component*). Kolejna składowa główna zdefiniowana jest jako pierwsza składowa główna otrzymana po odjęciu od wszystkich danych ich rzutu na pierwszą składową główną. Analogicznie, w sposób rekurencyjny definiuje się kolejne składowe główne. Po obliczeniu składowych głównych w liczbie odpowiadającej liczbie wymiarów, do których zredukowana ma być przestrzeń danych, wykonywane jest rzutowanie danych na podprzestrzeń rozpinaną przez te składowe główne.

Praktyczna metoda znajdowania składowych głównych, przedstawiona w [PCA], wykorzystuje macierz D, której kolumnami są wektory należące do zbioru danych. Znajdowany jest wektor wartości średnich **M**, który w następnym kroku odejmowany

jest od każdej z kolumn macierzy D. Następnie obliczana jest macierz kowariancji $C = D \cdot D^T$ i znajdowane są jej wektory własne. Wybrana liczba wektorów własnych o największych wartościach własnych tworzy bazę zredukowanej przestrzeni. Niech P oznacza macierz, której kolumny odpowiadają kolejnym wektorom tej bazy. Wówczas wektor danych **V** po przeniesieniu do przestrzeni zredukowanej przyjmuje postać $\mathbf{V}' = P^T \cdot (\mathbf{V} - \mathbf{M})$.

Liczba wymiarów zredukowanej przestrzeni może być ustalona z góry lub dopasowana do zbioru danych. Często liczbę tę wyznacza się w taki sposób, by uzyskać ustalony stosunek sumy wariancji wzdłuż wszystkich wymiarów do analogicznej wartości otrzymanej dla przestrzeni początkowej. Dla niektórych zbiorów danych redukcja tej wielkości o zaledwie 5% lub 10% umożliwia kilkukrotne zmniejszenie liczby wymiarów.

PCA stosowane jest w wielu dziedzinach, w których zachodzi konieczność obróbki danych o znacznej liczbie wymiarów, między innymi w analizie i rozpoznawaniu obrazów, gdzie często stosowane są też inne zbliżone techniki, np. dyskretna transformata kosinusowa, oraz w wizualizacji wielowymiarowych zbiorów danych w astronomii, geografii i szeroko rozumianej statystyce, gdzie zachodzi konieczność ich prezentacji w przestrzeni dwu- lub trójwymiarowej.

Wadę PCA stanowi fakt, że metoda ta optymalizuje bazę zredukowanej przestrzeni pod kątem rozróżnialności pojedynczych punktów, a nie całych klastrów. Pojedynczy punkt leżący z dala od wszystkich innych i nie wchodzący w skład żadnego klastra może powodować, że wariancja wzdłuż pewnej osi jest bardzo duża, a pomimo to rzutowanie na tę oś nie sprzyja późniejszemu odtworzeniu klastrów w postaci zbliżonej do oryginalnej.

Również w przypadku istnienia klastrów o kształcie silnie wydłużonym w jednym z kierunków, metoda PCA może znajdować przestrzeń zredukowaną, w której odtworzenie klastrów jest trudne lub niemożliwe pomimo istnienia lepszej redukcji do tej samej liczby wymiarów. Przykładowo, w sytuacji istnienia na płaszczyźnie dwóch klastrów składających się z punktów rozłożonych równomiernie wzdłuż dwóch równoległych odcinków o tej samej długości, metoda PCA jako składową główną wybierze kierunek równoległy do tych odcinków, co w przestrzeni zredukowanej do jednego wymiaru spowoduje zlanie się obu klastrów. Rozwiązaniem optymalnym jest w tym

przykładzie wybór kierunku głównego prostopadłego do owych odcinków. Powoduje on wprawdzie zlewanie się wielu punktów w jeden, ale za to umożliwia poprawne odróżnienie klastrów w przestrzeni o zredukowanym wymiarze. Niekorzystne dla klasteryzacji zachowanie PCA wynika ze wspomnianego wyżej faktu, że metoda ta optymalizuje rozróżnianie poszczególnych punktów, a nie klastrów.

W związku z wymienionymi wadami, zaproponowane zostały pewne rozszerzenia metody, m.in. ważone PCA i znormalizowane PCA (wprowadzone w [Koren]), które umożliwiają uzyskanie lepszego rozdzielenia klastrów w przestrzeni o zredukowanym wymiarze. PCA stanowi także pierwszy etap w metodzie ICA (*Independent Components Analysis*), służącej do znajdowania statystycznie niezależnych składowych sygnału i znajdującej zastosowanie również w przetwarzaniu języka naturalnego (por. uwagi o liniowej niezależności wektorów cech w punkcie 3.2).

3.4.3. SAI (uproszczone PCA)

Proponowana w niniejszym punkcie metoda SAI (SD AF IDF) stanowi połączenie metod PCA i TFIDF, umożliwiające redukcję wymiaru przestrzeni cech. Metoda ta jest specyficzna dla przestrzeni cech, nie nadaje się więc bezpośrednio do wykorzystania dla dowolnych danych jak metoda PCA.

Główny cel metody SAI stanowi redukcja wymiaru przestrzeni cech w ten sposób, aby w przestrzeni zredukowanej zachowane zostały w miarę możliwości klastry, i aby możliwe było ich rozdzielenie. Dla każdej cechy x stanowiącej wektor bazy przestrzeni cech (np. słowa lub n-gramu), obliczany jest współczynnik SAI określony wzorami:

- $SAI = SD \cdot AF \cdot (1 + IDF)$
- $SD = \begin{cases} SD', \text{gdy } SD' \neq 0 \\ AF^2, \text{gdy } SD' = 0 \end{cases}$
- $SD' = \sqrt{\frac{1}{P_x} \cdot \sum_{d \in D_x} \left(\frac{N_x(d)}{N(d)} - AF \right)^2}$
- $AF = \frac{1}{P_x} \cdot \sum_{d \in D_x} \frac{N_x(d)}{N(d)}$
- $IDF = \frac{P}{P_x}$,

gdzie D_x oznacza zbiór dokumentów zawierających cechę x, P_x moc tego zbioru, P liczbę wszystkich dokumentów, $N_x(d)$ liczbę wystąpień x w dokumencie d, a N(d) liczbę wszystkich cech odpowiedniego rodzaju w dokumencie d.

Baza przestrzeni o zredukowanym wymiarze jest tworzona z ustalonej liczby wektorów bazy początkowej o najwyższym współczynniku SAI. W odróżnieniu od metody PCA, SAI nie przekształca zatem wektorów bazy, a jedynie znajduje te, które należy uznać za najbardziej istotne. Podobieństwo do PCA polega na wykorzystaniu odchylenia standardowego do przypisania wag poszczególnym wektorom bazy, z tym, że zarówno odchylenie standardowe jak i wartość średnia częstości względnej są liczone jedynie dla dokumentów zawierających x. W przypadku gdy obliczona wartość odchylenia standardowego wynosi 0, zamiast niej stosowana jest wartość AF^2. Umożliwia to przypisanie niezerowego SAI słowom występującym tylko w pojedynczym dokumencie lub występującym z identyczną częstością w kilku dokumentach.

Współczynnik IDF zapobiega uzyskaniu zbyt wysokiego SAI przez słowa występujące we wszystkich lub prawie wszystkich dokumentach. Wykorzystanie tego czynnika o postaci podobnej do odpowiedniego członu wchodzącego w skład TFIDF powoduje, że metoda SAI przypisuje wysokie wagi słowom, które w niektórych dokumentach mogą osiągać wysokie wartości TFIDF, a więc dobrze nadawać się do oddzielenia tych dokumentów w procesie grupowania od innych, niepodobnych. SAI jest więc metodą w pewnym sensie „dopasowaną" do stosowania TFIDF jako wagi przy generacji współrzędnych – chociaż oczywiście można ją stosować również gdy używana jest inna funkcja ważąca.

Nieznaczną poprawę wyników algorytmu można czasem osiągnąć stosując zmodyfikowaną wersję SAI, wzbogaconą o dodatkowy człon:

- $SAI' = SAI \cdot (1 + \sqrt{GF})$,

gdzie GF jest globalną częstością względną cechy x. Rozszerzony wzór wymaga nieco większej ilości obliczeń oraz przechowywania w pamięci wartości GF dla poszczególnych cech, zatem charakteryzuje się gorszą wydajnością od podstawowej wersji SAI.

Redukcja wymiaru przestrzeni cech za pomocą metody SAI pozwala w większości przypadków na znaczne zmniejszenie wymiaru tej przestrzeni przy równoczesnym zachowaniu stosunkowo dużej liczby niezmienionych klastrów. Wadę tej metody stanowi jednak fakt, że jest ona mniej elastyczna od PCA, gdyż nie jest możliwe jej bezpośrednie wykorzystanie do redukcji wymiaru dowolnych zbiorów danych.

4. Grupowanie

Zagadnienie grupowania (klasteryzacji) polega na podziale zadanego zbioru dokumentów tekstowych na pewną liczbę grup (klastrów), w ramach których dokumenty charakteryzują się podobną treścią. Podobieństwo rozumiane jest najczęściej jako podobieństwo tematyki dokumentów, ale możliwe jest także grupowanie według innych kryteriów, np. według pewnych cech stylu. Pożądane jest, aby dokumenty przydzielone do tej samej grupy były do siebie wzajemnie jak najbardziej podobne, zaś dokumenty przydzielone do różnych grup powinny się między sobą jak najbardziej różnić.

Zastosowania praktyczne algorytmów grupowania dokumentów obejmują głównie współpracę z różnego rodzaju wyszukiwarkami informacji. Liczba znalezionych dokumentów pasujących do poszukiwanego hasła może być bardzo duża, ale odpowiednie uporządkowanie wyników pozwala na łatwiejsze znalezienie wśród nich dokumentów rzeczywiście interesujących użytkownika. Szczególnie duże znaczenie może mieć grupowanie w sytuacji, gdy hasło posiada w zależności od kontekstu różne znaczenia i ustalenie, które z nich użytkownik miał na myśli nie jest możliwe na podstawie samego tylko zapytania. W takiej sytuacji wyniki są prezentowane w postaci pogrupowanej, co pozwala od razu odrzucić część z nich jako dotyczących innego tematu, niż interesujący użytkownika. Grupowanie może pomóc w znalezieniu właściwych dokumentów również wtedy, gdy zapytanie ma jednoznaczny sens, gdyż grupy wprowadzają dodatkowy porządek w wynikach i ułatwiają wybór bardziej szczegółowej kategorii dokumentów do przeszukania. Przykładem wykorzystania algorytmów grupowania do dokumentów wyszukiwanych w sieci WWW za pomocą wyszukiwarek internetowych jest projekt Carrot2, którego podstawę stanowi algorytm LINGO, opisany w [LINGO].

Algorytmy grupowania znajdują także wiele zastosowań w odniesieniu do danych innych niż wektory reprezentujące dokumenty tekstowe. Szczególnie ważną rolę odgrywają w biologii, gdzie są wykorzystywane do analizy genów oraz tworzących je nukleotydów. Do najczęstszych zastosowań praktycznych klasteryzacji należy przetwarzanie informacji znajdujących się w różnego rodzajach bazach danych, m.in. w katalogach bibliotecznych, geologicznych bazach danych wykorzystywanych przy poszukiwaniu złóż ropy naftowej oraz w bazach danych zawierających informacje o klientach, które są stosowane w systemach CRM.

4.1. Algorytmy dzielące i hierarchiczne

Wyróżnia się dwie główne klasy algorytmów grupowania: dzielące i hierarchiczne ([Jain]). Algorytmy dzielące zwracają w wyniku swojego działania pojedynczy podział, zwykle stanowiący lokalne ekstremum pewnej funkcji celu. Ponieważ ustalenie z góry właściwej liczby grup może być zadaniem trudnym, w wielu przypadkach algorytmy takie wymagają podania jej jako jednego z parametrów.

Problem wyboru liczby grup w pewnym stopniu rozwiązują algorytmy hierarchiczne. Jako wynik zwracają one całe drzewo grup, w którym grupy większe podzielone są stopniowo na coraz mniejsze podgrupy. Pozwala to na wybór najlepszego podziału, a więc i przypisanej do niego liczby grup, spośród pewnej liczby podziałów reprezentowanych przez drzewo. Wybór może nastąpić już po zakończeniu działania algorytmu, co jest szczególnie użyteczne wtedy, gdy kryteria oceny jakości podziałów mogą ulegać częstym zmianom (na przykład w wyniku interakcji z użytkownikiem). Każdy hierarchiczny algorytm grupowania może zatem łatwo zostać wykorzystany do stworzenia opartego na nim algorytmu dzielącego: wystarczy znaleźć kryterium określające, na którym poziomie drzewa podziałów należy algorytm zatrzymać i zwrócić odpowiedni podział jako rozwiązanie „optymalne”.

4.2. Algorytmy oparte o centroidy i oparte o gęstość danych

Inne kryterium podziału algorytmów grupowania stanowi sposób przydziału punktów do poszczególnych klastrów. Większość klasycznych algorytmów, np. k-means, reprezentuje klaster przez jego centroid, czyli punkt wyznaczany jako geometryczny środek punktów należących do klastra. Punkt ten służy następnie jako „centrum” przyciągające pozostałe punkty klastra. Wadę tego podejścia stanowi fakt, że zwykle znajdowane mogą być jedynie klastry o kształtach w przybliżeniu kulistych i dobrze od siebie oddzielone.

Niektóre algorytmy nie posługują się pojęciem centroidów, a zamiast tego operują pojęciem gęstości danych. Gęstość danych wokół punktu **p** definiuje się jako liczbę punktów leżących od niego w odległości nie większej niż ε. Zbiór takich punktów oznaczany jest $N_\varepsilon(\mathbf{p})$. Punkt **p**, dla którego moc tego zbioru (czyli gęstość danych wokół niego) jest nie mniejsza od pewnej wartości granicznej m nazywamy punktem wewnętrznym. Punkty inne niż wewnętrzne nazywamy punktami brzegowymi. Punkt **q** nazywamy bezpośrednio osiągalnym z punktu **p** jeżeli należy on do $N_\varepsilon(\mathbf{p})$ i **p** jest

punktem wewnętrznym. Punkt **q** nazywamy osiągalnym z punktu **p** jeżeli istnieje taki ciąg punktów, którego pierwszym elementem jest **p**, ostatnim **q**, a każdy kolejny punkt jest bezpośrednio osiągalny z poprzedniego. Punkty **p** i **q** nazywamy połączonymi jeżeli istnieje punkt **r**, z którego zarówno **p** jak i **q** są osiągalne. Klastrem nazywamy każdy maksymalny zbiór punktów połączonych. Każdy punkt wewnętrzny jednoznacznie wyznacza klaster, do którego należy. Metody grupowania oparte na gęstości danych często pozwalają na wykrycie grup o skomplikowanym kształcie, co często nie jest możliwe przy wykorzystaniu metod opartych na centrach klastrów.

4.3. Przykłady algorytmów

4.3.1. K-means

K-means stanowi klasyczny i bardzo popularny algorytm grupowania oparty o centroidy, opisany m.in. w [Späth] i [Mirkin]. Początkowo tworzona jest pewna liczba klastrów, ustalana na podstawie heurystyki lub podawana przez użytkownika jako parametr. Położenia ich centroidów ustalane są losowo. W kolejnych iteracjach każdy punkt przypisywany jest do tego klastra, którego centroid jest mu najbliższy w pewnej ustalonej wcześniej metryce, a po przypisaniu punktów do klastrów obliczane jest nowe położenie każdego centroidu (jego współrzędne są obliczane jako średnie arytmetyczne współrzędnych punktów należących do klastra lub jako ich środek ciężkości). Warunkiem stopu jest brak zmiany przypisania punktów do klastrów w kolejnych iteracjach. Zwykle klastry puste są usuwane pod koniec działania algorytmu.

Prostota algorytmu i jego duża wydajność uczyniły go popularnym pomimo licznych wad. Losowe ustalenie położeń centroidów klastrów w pierwszej iteracji skutkuje brakiem powtarzalności wyników działania algorytmu dla tych samych danych wejściowych. Konieczne jest też ustalenie z góry maksymalnej liczby klastrów. Podobnie jak inne metody oparte o centra klastrów, również k-means jest w stanie znajdować jedynie klastry o kształcie zbliżonym do kulistego.

4.3.2. HWH

Klasteryzator HWH autorstwa Grzegorza Kolendo znajduje klastry symulując zachowanie oddziałujących ze sobą cząstek. Grupowanym wektorom odpowiadają nieruchome cząstki, do których w symulacji przyciągane są centroidy klastrów. Aby zapobiec nachodzeniu na siebie klastrów, centroidy oddziałują na siebie nawzajem siłami

odpychającymi. Po ustabilizowaniu się pozycji centroidów, każdy wektor jest przypisywany do centroidu leżącego najbliżej.

Dostępna w czasie pisania niniejszej pracy wersja algorytmu odbiegała od wersji ostatecznej i charakteryzowała się gorszą od niej jakością klasteryzacji oraz wydajnością. Planowane jest porównanie wyników uzyskanych z użyciem obu wersji w późniejszym terminie.

4.3.3. DBSCAN

Algorytm DBSCAN zaproponowany w [Ester] znajduje klastry w oparciu o gęstość danych. Rozpoczyna on działanie od losowo wybranego punktu **p**. Jeżeli punkt ten jest punktem wewnętrznym, tworzony jest składający się tylko z tego punktu klaster, a następnie dołączane są do niego wszystkie punkty osiągalne z **p**. Po wyczerpaniu wszystkich takich punktów, wybierany jest losowo pewien nieodwiedzony jeszcze punkt i rozpoczynana jest kolejna iteracja. Jeżeli **p** jest punktem brzegowym, nie jest on dołączany do żadnego klastra. W odróżnieniu od większości algorytmów opartych o centroidy, DBSCAN traktuje niektóre punkty jako szum, który nie powinien zostać przypisany do żadnej ze znajdowanych grup. Algorytm kończy działanie gdy wszystkie punkty zostały już odwiedzone.

Parametrami wpływającymi na sposób działania DBSCAN są: promień sąsiedztwa uwzględnianego przy obliczaniu gęstości danych oraz minimalna gęstość danych potrzebna do uznania punktu za punkt wewnętrzny. Parametry te mogą być ustalone z góry lub mieć przypisane pewne wartości dopasowywane automatycznie do grupowanego zbioru punktów. Mogą one również zostać dobrane na podstawie pewnych cech klastrów znalezionych we wstępnym przebiegu algorytmu DBSCAN, po którym dopiero jest wykonywany przebieg właściwy, korzystający już z dopasowanych parametrów.

4.4. Nadawanie klastrom nazw

W celu zwiększenia przejrzystości wygenerowanego podziału zbioru dokumentów na klastry, klastrom tym nadaje się często etykiety charakteryzujące zawarte w nich dokumenty. Nadawanie grupom dokumentów nazw czytelnych dla człowieka jest szczególnie ważne wtedy, gdy grupowanie wykorzystywane jest do porządkowania wyników wyszukiwania, np. w wyszukiwarkach internetowych. Trafny dobór ozna-

czeń poszczególnych grup pozwala użytkownikowi na zaoszczędzenie czasu i szybką ocenę przydatności dla jego celów poszczególnych zestawów wyników.

Gdy używane są algorytmy grupowania oparte o centroidy, zwykle centroid stanowi wektor charakteryzujący cały klaster i tylko jego współrzędne są wykorzystywane w celu uzyskania tekstowego opisu grupy dokumentów. Innym podejściem, działającym także w przypadku użycia algorytmów grupowania opartych o gęstość danych, może być wybór pewnego wektora należącego do klastra jako reprezentanta całej grupy. Wybór ten zwykle odbywa się na podstawie pewnej heurystyki, np. wybierany może być wektor leżący najbliżej geometrycznego środka klastra (medoid). W przypadku algorytmów typu DBSCAN, zdolnych do znajdowania klastrów o skomplikowanym kształcie, takie rozwiązanie może nie być satysfakcjonujące, gdyż geometryczny środek klastra wcale nie musi leżeć blisko punktów do tego klastra przypisanych ([Jain]). Można wówczas posłużyć się metodami korzystającymi z pojęcia gęstości danych i np. za reprezentanta klastra uznać dokument, wokół którego gęstość ta jest największa.

Po znalezieniu wektora reprezentującego klaster, znajdowane są wyróżniające go spośród innych współrzędne. Najprostszą metodę stanowi wybór jednej lub więcej współrzędnych, których wartość jest największa i utworzenie nazwy grupy na podstawie cech, którym w przestrzeni cech odpowiadają te współrzędne. W innym podejściu, znajdowany jest geometryczny środek zbioru punktów odpowiadających wszystkim klastrom poza aktualnie rozpatrywanym i obliczana jest różnica tego wektora oraz wektora reprezentującego bieżący klaster. Dopiero na podstawie maksymalnych współrzędnych tej różnicy wybierana jest nazwa. Rozwiązanie to posiada istotną przewagę nad opisanym wcześniej prostszym modelem, gdyż jako składowe nazwy wybiera przede wszystkim cechy, których obecność w dokumentach klastra zdecydowanie różni się od ich zawartości w pozostałych dokumentach. W ten sposób zmniejsza się prawdopodobieństwo zajścia sytuacji, w której wiele klastrów otrzymałoby podobne lub wręcz identyczne nazwy.

Bardziej złożone metody nie tworzą nazw grup bezpośrednio przez sklejenie ze sobą nazw wybranych cech, lecz generują nazwy bardziej przypominające zwykłe zdania lub ich fragmenty. Interesujące podejście wykorzystywane jest w algorytmie LINGO ([LINGO]), w którym najpierw znajdowane są „dobre”, opisowe nazwy grup, a

dopiero później na ich podstawie wykonywana jest klasteryzacja. Znajdowanie etykiet grup wykonywane jest poprzez znalezienie za pomocą LSA „abstrakcyjnych idei" wyrażonych w dokumentach, a następnie dopasowanie do nich często występujących fraz oraz wyeliminowanie z tak utworzonej listy powtarzających się wyrażeń. Na podstawie znalezionego zbioru charakterystycznych fraz, stanowiących etykiety grup, wykonywana jest klasyfikacja dokumentów i przypisanie ich do grup związanych z poszczególnymi opisami. Znalezione etykiety definiują z góry położenie klastrów, dzięki czemu możliwe jest sprowadzenie zagadnienia klasteryzacji do zagadnienia klasyfikacji.

5. Implementacja

Niniejszy rozdział przedstawia zaimplementowane w programie algorytmy, sposoby współpracy poszczególnych modułów oraz informacje o plikach odczytywanych i zapisywanych przez aplikację. Szczegóły takie jak nazwy metod zawartych w poszczególnych klasach są dokładnie opisane bezpośrednio w kodzie (komentarze JavaDoc) i poza uzasadnionymi wyjątkami nie zostały tutaj zamieszczone.

5.1. Środowisko programistyczne

Do implementacji algorytmów wybrano język Java oraz środowisko programistyczne Eclipse. Program tworzono i testowano na maszynie wirtualnej Javy zawartej w Sun JDK 1.4.2_04, działającej w systemie operacyjnym GNU/Linux. Język Java został wybrany ze względu na jego przenośność oraz na łatwość zarządzania pamięcią w programie.

Do tworzenia zestawów danych testowych i analizy wyników testów wykorzystano skrypty powłoki sh oraz skrypty w językach awk i sed.

5.2. Baza danych konfiguracyjnych

Opcje konfiguracyjne programu są dostępne za pośrednictwem singletonowego obiektu klasy `Config`. Mają one postać par klucz-wartość, przy czym klucz jest łańcuchem klasy `String`, natomiast wartość może być obiektem dowolnego typu. Klasa `Config` posiada metody automatycznie opakowujące typy proste w ich obiektowe odpowiedniki, dzięki czemu możliwy jest bezpośredni odczyt i zapis zmiennych tych typów.

Stan bazy danych konfiguracyjnych jest pomiędzy uruchomieniami programu przechowywany w pliku tekstowym `mgr.conf` (kodowanie UTF-8). Aby wartości niestandardowych typów obiektowych mogły być zapisywane do pliku i z niego odczytywane, odpowiednie klasy muszą implementować interfejs `Config.ConfigSavable` oraz posiadać konstruktor bezparametrowy.

Możliwe jest zdefiniowanie niektórych kluczy jako ulotnych, co powoduje, że odpowiadające im wartości nie są przechowywane pomiędzy uruchomieniami programu.

Format pliku konfiguracyjnego jest następujący:

```
łańcuch      := dowolny ciąg znaków zakończony znakiem nowego wiersza
łańcuchEsc   := jak łańcuch, ale wszystkie wystąpienia
```

```
                "\\n" odczytujemy jako znak nowego wiersza, zaś
                "\\x" gdzie x oznacza dowolny znak różny od 'n'
                traktujemy jako literalny znak 'x'
klucz         := łańcuchEsc
nazwa_klasy   := łańcuch
wartość       := łańcuchEsc
pusty_wiersz  := '\n'
cały_plik     := (pusty_wiersz* klucz pusty_wiersz*
                  nazwa_klasy wartosc)*
```

5.3. Formaty danych wejściowych

Opisana w punkcie 5.5.1.4 klasa `TextDocument`, reprezentująca zawartość dokumentu tekstowego, może odczytywać dokumenty w trzech formatach: tekstowym, HTML oraz XML. Funkcje realizujące wczytywanie tych danych obsługują nie tylko pliki znajdujące się na lokalnych dyskach, lecz potrafią odczytywać dane z dowolnego strumienia dostępnego za pośrednictwem klas implementujących interfejs `Reader` lub (w przypadku formatu XML) `InputSource`. Dzięki temu dodanie w razie potrzeby możliwości pobierania dokumentów bezpośrednio z sieci WWW lub innego nie obsługiwanego obecnie źródła jest bardzo proste.

5.3.1. Zwykły tekst

Zwykły tekst stanowi najprostszy z możliwych sposobów reprezentacji dokumentów tekstowych. Poza samą treścią dokumentu – zdaniami i wyrazami – format ten nie dostarcza żadnych dodatkowych informacji mogących ułatwić proces grupowania. Wczytywany tekst jest w programie interpretowany zgodnie z kodowaniem określonym zmienną konfiguracyjną `files.text.charset`.

5.3.2. HTML

W programie wykorzystano parser języka HTML wbudowany w bibliotekę Swing. Wybranym znacznikom HTML przypisano wagi wyrażające istotność tekstu w nich zawartego. Wagi poszczególnych słów są obliczane jako iloczyn wag znaczników, wewnątrz których te słowa się znajdują. Dotyczy to zarówno znaczników występujących w ciele dokumentu jak i w nagłówku. Pewne znaczniki (np. `P`) powodują, że tekst w nich zawarty jest traktowany jako osobne zdanie nawet wtedy, gdy znaki interpunkcyjne na to nie wskazują (taka interpretacja wynika jednak jednoznacznie ze sposobu stosowania tych znaczników).

Program potrafi wydobyć listę słów kluczowych z treści znacznika `META KEYWORDS` oraz opis dokumentu ze znacznika `META DESCRIPTION` i jego tytuł z `TITLE`. Słowa wydobyte z nagłówka są wstawiane na początek wewnętrznej reprezentacji treści dokumentu w programie. Do treści dokumentu są również włączane opisy plików graficznych umieszczonych w dokumencie (atrybuty `ALT` i `TITLE` znacznika `IMG`). Program honoruje kodowanie znaków podane w znaczniku `META HTTP-EQUIV` oraz język dokumentu wpisany w znaczniku `META LANGUAGE`.

5.3.3. XML z rozbiorem gramatycznym dokumentu

Informacje o budowie gramatycznej tekstu mogą w istotny sposób ułatwić porównywanie go z innymi dokumentami. Program nie posiada możliwości analizy tekstu pod kątem jego gramatyki, ale może takie informacje wykorzystać, gdy otrzyma je z zewnętrznego źródła. Do przekazywania tych informacji służy specjalny format XML.

Przykładowy plik XML reprezentujący dokument o treści *Ala ma kota.* przedstawiono poniżej:

```
<?xml version="1.0" encoding="utf-8"?>
<dokument jezyk="pl">
      <zdanie nr="1" rodzaj="oznajmujace" zlozone="nie">
            <token id="1" lemat="Ala" czescmowy="rzeczownik"
                  liczba="pojedyncza" czesczdania="podmiot"
                  slowo="Ala"/>
            <token id="2" lemat="mieć" czescmowy="czasownik"
                  liczba="pojedyncza" czesczdania="orzeczenie"
                  slowo="ma"/>
            <token id="3" lemat="kot" czescmowy="rzeczownik"
                  przypadek="dopelniacz" rodzaj="meski"
                  liczba="pojedyncza" czesczdania="dopelnienie"
                  slowo="kota" waga="2.0"/>
      </zdanie>
</dokument>
```

Znacznikiem zawierającym w sobie całą treść dokumentu jest znacznik `dokument`, posiadający jeden opcjonalny atrybut `jezyk` umożliwiający podanie informacji o języku tekstu. We wnętrzu tego znacznika może znajdować się dowolna liczba znaczników typu `zdanie`, posiadających opcjonalne atrybuty: `nr` (identyfikator; obecnie jest on ignorowany), `rodzaj` (`oznajmujace`, `pytajace`, `rozkazujace`, `inne`) oraz `zlozone` (czy zdanie jest zdaniem złożonym: `tak`, `nie`).

Wewnątrz znaczników reprezentujących zdania znajdują się znaczniki `token` reprezentujące słowa wchodzące w skład zdania. Bardziej ogólną nazwę `token` wybra-

no zamiast narzucającej się nazwy `slowo`, aby była ona nadal aktualna w razie rozszerzenia formatu o obsługę innych elementów, np. istotnych znaków przestankowych, wyróżnionych symboli matematycznych itd. Atrybuty znacznika token obejmują m.in.: `id` (identyfikator; obecnie ignorowany), `lemat`, `czescmowy` (`rzeczownik`, `czasownik`, `przymiotnik`, `przyslowek`, `spojnik`, `inna`), `czesczdania` (`podmiot`, `orzeczenie`, `przydawka`, `okolicznik`, `dopelnienie`, `inna`), `liczba` (`pojedyncza`, `mnoga`), `slowo`, `waga`. Atrybut `slowo` jest obowiązkowy, zaś wszystkie pozostałe są opcjonalne. Znaczenie tych atrybutów pokrywa się ze znaczeniem odpowiednich pól klasy `Word`, opisanej w punkcie 5.5.1.1.

Zarówno w przypadku znacznika `zdanie` jak i `token`, atrybuty nie wykorzystywane przez program (w powyższym przykładzie: `przypadek` i `rodzaj` w słowie o `id` równym 3) są ignorowane.

Analiza dokumentu XML jest dokonywana przy pomocy niewalidującego parsera z pakietu `javax.xml.parsers`, który na podstawie strumienia danych XML tworzy obiektowy model dokumentu (*Document Object Model*, DOM). Z reprezentacji DOM odczytywane są znaczniki i związane z nimi wartości atrybutów, które są następnie zamieniane na odpowiednie struktury w klasie `TextDocument`.

5.4. Zarządzanie pamięcią

Ponieważ przetwarzane zbiory dokumentów tekstowych mogą zawierać znaczną ich liczbę, aplikacja korzysta z klas, których celem jest umożliwienie przechowywania i przetwarzania danych, których rozmiar przekracza rozmiar dostępnej pamięci wirtualnej.

Klasa `DiskDumpMap` implementuje interfejs `Map`. Służy ona do przechowywania odwzorowań (map), które posiadają stosunkowo nieliczny zbiór kluczy, ale za to zbiór wartości będących obiektami o potencjalnie znacznych wymaganiach pamięciowych. Klasa korzysta z `WeakHashMap` – mapy, której klucze są przechowywane w słabych referencjach (*weak references*) języka Java. Równocześnie, obiekty są za pośrednictwem serializacji składowane w plikach tymczasowych na dysku. Obiekty, do których odwołują się jedynie słabe referencje, mogą zostać sfinalizowane przez odśmiecacz pamięci, a przypisana im pamięć – przeznaczona dla innych obiektów. Obiekt typu `DiskDumpMap` obsługuje żądanie zwrócenia wartości przypisanej do pewnego klucza sprawdzając najpierw, czy odpowiednia wartość znajduje się w `WeakHashMap`. Jeżeli

obiekt był niedawno wykorzystywany, to ma sporą szansę wciąż znajdować się w pamięci, skąd może zostać pobrany natychmiast. Jeżeli obiekt nie posiada odwzorowania w `WeakHashMap`, wartość odpowiadająca kluczowi ładowana jest z dysku. Zbiór kluczy przechowywany jest w pamięci przez cały czas, tak więc klasa ta nie chroni całkowicie przed wyczerpaniem dostępnej pamięci, lecz jedynie opóźnia moment jego wystąpienia.

Drugą klasą umożliwiającą przechowywanie znacznych ilości danych jest `BufferedCounterMap`. Służy ona do zliczania wystąpień obiektów, w sytuacji gdy kluczy może być bardzo wiele. W pamięci przechowywana jest zwykła mapa typu `HashMap`, w której każdemu kluczowi przypisany jest licznik (obiekt typu `Counter`). Mapa zapełniana jest aż do osiągnięcia ustalonego z góry rozmiaru (wyrażonego przez liczbę przechowywanych odwzorowań). Po zapełnieniu mapy, zawarte w niej odwzorowania są zapisywane w pliku tymczasowym (za pomocą serializacji), zaś sama mapa jest opróżniana i wykorzystywana ponownie. Klasa udostępnia metodę `increaseCounter()`, pozwalającą na stworzenie nowego licznika w sytuacji gdy mapa nie zawiera podanego klucza lub zwiększenie licznika gdy klucz istnieje. Jeżeli ten sam licznik jest zwiększany kilkukrotnie, a w międzyczasie zawartość mapy zostaje zapisana na dysku, to kolejne wartości licznika (oznaczające w rzeczywistości przyrosty) są zapisywane w pliku jako niezależne obiekty. Odzyskiwanie danych z mapy odbywa się za pomocą metody `getMaxValues()`. Metoda ta posługuje się algorytmem sortowania zewnętrznego przez scalanie opisanym w [Knuth], aby na początku posortować według kluczy wszystkie dane, przechowywane zarówno w pamięci jak i na dysku, co pozwala na dodanie do siebie wartości różnych liczników należących do tego samego klucza. W następnym kroku dane są sortowane według wartości zsumowanych liczników (również metodą sortowania zewnętrznego), dzięki czemu możliwe jest znalezienie kluczy, do których przypisane są liczniki o największych wartościach. Metoda zwraca podaną jako parametr liczbę par klucz-licznik, dla których licznik jest największy. Zużycie pamięci podczas całej operacji nie przekracza nigdy w istotny sposób limitu ustalonego podczas konstrukcji obiektu.

Wymienione klasy wykorzystywane są przede wszystkim w klasach generatorów współrzędnych, które zbierają informacje statystyczne o wielu dokumentach i często generują znaczne ilości danych tymczasowych używanych podczas obliczeń. Ponieważ klasy te korzystają z dysku jako pamięci zewnętrznej, ich działanie jest dla zbiorów

danych, które mogą być przetworzone w całości z użyciem tylko pamięci operacyjnej znacznie wolniejsze niż klas implementujących te same algorytmy, ale korzystających ze zwykłych map przechowywanych cały czas w RAM.

5.5. Struktura programu

Z wyjątkiem kilku klas pomocniczych z projektów Egothor i Stempel, umieszczonych w pakiecie `org`, wszystkie klasy należące do programu zostały zgrupowane w pakiecie `edu.mgr`. Bezpośrednio w nim umieszczono klasy związane z obsługą interfejsu użytkownika. Klasy realizujące zadania obliczeniowe rozdzielono pomiędzy pięć podpakietów: `base` (podstawowe struktury danych), `clusterizer` (klasteryzatory i klasy z nimi ściśle związane), `converter` (generatory współrzędnych), `filter` (filtry wstępnego przetwarzania tekstu) i `util` (klasy pomocnicze). Katalogi `edu` oraz `org`, zawierające wszystkie pliki z kodem źródłowym w Javie (kodowanie UTF-8) i skompilowane pliki `.class`, znajdują się w głównym katalogu projektu.

Dla klas i zmiennych użyte zostały w kodzie nazwy angielskie (wyjątek stanowią nazwy odnoszące się do gramatyki języka polskiego), natomiast komentarze i interfejs użytkownika napisane są w języku polskim.

Wszelkie pliki pomocnicze wykorzystywane przez filtry, np. pliki zawierające stop-listy dla różnych języków, znajdują się w odpowiadających nazwom klas je wykorzystujących podkatalogach katalogu `data`. Pliki zawierające dane dla poszczególnych języków noszą nazwy odpowiadające dwuliterowym kodom ISO tych języków (np. `pl` dla języka polskiego). Pliki wykorzystywane przez Eclipse oraz plik konfiguracyjny programu umieszczone zostały w katalogu głównym. Tam znajdują się także skrypty ułatwiające uruchamianie programu oraz skrypty używane do generacji danych testowych i testowania programu.

5.5.1. Podstawowe struktury danych

Podstawowe struktury danych używane do reprezentacji dokumentów zgromadzone zostały w pakiecie `edu.mgr.base`. Pomimo że wszystkie wykorzystane w programie algorytmy wykorzystują podejście *bag of words*, struktury danych zostały zaprojektowane z myślą o elastyczności i zachowują informacje o porządku zdań i słów w dokumencie. Informacje te mogą z łatwością być wykorzystane przez nowe klasy dodawane do aplikacji.

5.5.1.1. Word

Klasa `Word` przechowuje informacje o pojedynczym słowie wchodzącym w skład dokumentu. Obejmują one: treść słowa w brzmieniu w jakim występuje ono w dokumencie, lemat, część mowy i zdania, liczbę oraz wagę.

Zarówno słowo jak i lemat są przechowywane w zmiennych typu `String` co automatycznie powoduje, że obsługiwany jest podzestaw BMP znaków Unicode. Dzięki temu możliwe jest przechowywanie znaków diakrytycznych większości języków oraz znaków alfabetów innych niż łaciński.

Waga jest liczbą typu `double` wyrażającą względne znaczenie konkretnego wystąpienia słowa w danym dokumencie. Wynosi ona domyślnie 1.0 i może być zwiększona lub zmniejszona poprzez podanie innej wartości w pliku XML, zastosowanie odpowiednich znaczników w pliku HTML lub poprzez działanie filtrów. W przypadku stosowania podejścia *bag of words*, wykorzystanie wagi podczas porównywania dokumentów polega na zwiększaniu licznika wystąpień napotkanego słowa o tę właśnie liczbę zamiast o 1 podczas zliczania słów, co skutkuje przypisaniem większego znaczenia słowom o większej wadze.

5.5.1.2. NGram

Do reprezentacji w programie n-gramów służy klasa `NGram`. Nie jest ona wykorzystywana do przechowywania treści dokumentu, lecz korzystają z niej różne algorytmy tworzące jej instancje jako zmienne tymczasowe.

Konstruktor pozwala wybrać rodzaj n-gramu (uporządkowany bądź nieuporządkowany) oraz przewidywaną wartość n. N-gramy uporządkowane przechowują wchodzące w ich skład słowa w liście, zaś n-gramy nieuporządkowane wykorzystują w tym celu zbiór z powtórzeniami (reprezentowany przez mapę słowo → liczba wystąpień). Liczba słów wchodzących w skład n-gramu może ulegać zmianie, ale ze względu na wydajność najbardziej wskazane jest nieprzekraczanie rozmiaru podanego w konstruktorze.

Słowa reprezentowane są przez zwykłe łańcuchy – dodatkowe informacje przechowywane w klasie `Word` nie są zachowywane. N-gramowi przypisana może być waga (przechowywana poza obiektem), której znaczenie jest podobne do znaczenia wagi pojedynczego słowa. Sposób obliczania wagi n-gramu na podstawie wagi wcho-

dzących w jego skład słów nie jest zdefiniowany w samej klasie `NGram` i zależy od sposobu jej wykorzystania przez poszczególne algorytmy. W algorytmach wykorzystywanych w programie przyjęto konwencję, że wagą n-gramu jest iloczyn wag słów wchodzących w jego skład.

Zliczanie n-gramów w dokumencie wykonywane jest w taki sposób, że w skład tego samego n-gramu wchodzić mogą jedynie słowa należące do tego samego zdania.

5.5.1.3. Sentence

Klasa `Sentence` reprezentuje w programie zdania, z których składają się dokumenty tekstowe. Zawiera ona pola z informacjami o rodzaju zdania (czy jest ono zdaniem oznajmującym, pytającym, rozkazującym czy innym (nieznanym)) oraz o tym, czy zdanie jest zdaniem złożonym, czy prostym. Słowa wchodzące w skład zdania (obiekty typu `Word`) są przechowywane w liście odzwierciedlającej ich kolejność w zdaniu.

Oprócz metod umożliwiających dostęp do danych obiektu, klasa `Sentence` posiada metody pozwalające na utworzenie swojej listy słów na podstawie łańcucha tekstowego. W klasie zawarta jest rozbudowana lista znaków uznawanych za separatory słów, obejmująca oprócz znaków ASCII także liczne unikodowe znaki interpunkcyjne, coraz częściej spotykane w dokumentach tekstowych dostępnych w internecie.

5.5.1.4. TextDocument

`TextDocument` to klasa reprezentująca dokument tekstowy. Treść dokumentu jest reprezentowana przez listę zdań (obiektów typu `Sentence`) wchodzących w jego skład oraz łańcuch identyfikujący język dokumentu. Każdy obiekt klasy `TextDocument` posiada unikatowy dla danej instancji programu identyfikator, który jest generowany automatycznie podczas tworzenia instancji klasy. Identyfikator ten jest wykorzystywany do późniejszego skojarzenia z dokumentem wektora współrzędnych reprezentującego ten dokument.

Metody klasy pozwalają m.in. na dostęp do zdań wchodzących w skład dokumentu oraz na przeglądanie za pomocą specjalnego iteratora kolejno wszystkich jego słów bez uwzględniania granicy zdań.

Klasa `TextDocument` zawiera metody pozwalające na wczytywanie treści dokumentów z plików dyskowych lub innych strumieni danych. Obsługiwane formaty (zwykły tekst, HTML i XML) opisane zostały szczegółowo w punkcie 5.3. Klasa za-

wiera także zbiór znaków separatorów zdań analogiczny do zawartego w klasie `Sentence` zbioru separatorów słów.

Przechowywanie treści dokumentu w postaci listy zdań zwiększa elastyczność klasy, ale odbywa się to kosztem wydajności w sytuacji, gdy wykorzystywane jest podejście *bag of words*. W przypadku jego stosowania, bardziej wydajne byłoby wykonywanie wstępnego przetwarzania tekstu już na etapie jego wczytywania z pliku, a następnie przechowywanie dokumentu w postaci mapy odwzorowującej słowa lub n-gramy na liczbę ich wystąpień w tekście. Przy wykorzystaniu bardziej elastycznego, ale nie zoptymalizowanego dla przypadku *bag of words* rozwiązania, obliczenie liczby wystąpień poszczególnych słów wymaga przejrzenia całej listy słów przechowywanych w obiekcie i dopiero na ich podstawie wygenerowania mapy opisanej powyżej.

5.5.1.5. Vector

Klasa `Vector` służy do przechowywania i manipulowania dowolnie długich wektorów liczb typu `double`. Sposób przechowywania danych umożliwia efektywne operowanie wektorami rzadkimi, to znaczy takimi, których większość współrzędnych ma wartość zero.

Publiczne pole `id` służy do przechowywania identyfikatora dokumentu, którego reprezentacją jest dany wektor. Taki sposób kojarzenia wektorów z dokumentami został wybrany zamiast bardziej obiektowo zorientowanego przechowywania referencji do odpowiedniego obiektu typu `TextDocument` z tego względu, że umożliwia on usunięcie z pamięci obiektu dokumentu bez utraty informacji o tym, któremu dokumentowi odpowiada który wektor, a co za tym idzie, ogranicza zapotrzebowanie programu na pamięć RAM. Przed utworzeniem wektorów program zapamiętuje mapę odwzorowującą identyfikatory dokumentów na nazwy plików, zaś same obiekty zawierające treść tych dokumentów mogą zostać usunięte z pamięci zaraz po utworzeniu wektorów.

5.5.1.6. Cluster

`Cluster` to klasa reprezentująca klaster punktów, wykorzystywana w procesie grupowania. Przechowuje ona zbiór wektorów należących do klastra oraz wyróżniony wektor stanowiący jego środek.

5.5.2. System wtyczek

Ponieważ elastyczność stanowiła jedno z głównych założeń rozwijanego programu, stało się konieczne stworzenie architektury umożliwiającej dodawanie nowych algorytmów do już obsługiwanych oraz pozwalającej na swobodny wybór dowolnego algorytmu dla każdego zadania realizowanego podczas procesu grupowania dokumentów. Założenia te zostały zrealizowane za pośrednictwem systemu wtyczek dynamicznie ładowanych podczas działania aplikacji.

Wtyczki dostarczają funkcji pozwalających określić dla nich liczbę, nazwy i typy parametrów wpływających na sposób ich działania. Na podstawie informacji o dostępnych wtyczkach automatycznie tworzony jest interfejs użytkownika.

Klasy wtyczek zostały podzielone na cztery kategorie, w zależności od wykonywanych przez nie zadań. Kategoriami tymi są: filtry, realizujące różne metody wstępnego przetwarzania dokumentu, generatory współrzędnych (konwertery), generujące reprezentacje dokumentów w przestrzeniach cech, metryki, realizujące funkcje obliczania odległości w przestrzeniach liniowych oraz klasteryzatory, realizujące proces grupowania wektorów.

Wynajdowanie dostępnych wtyczek poszczególnych kategorii odbywa się poprzez przejrzenie katalogów dyskowych przeznaczonych dla każdej z nich w poszukiwaniu plików klas Javy (`*.class`). Za wtyczki są uznawane klasy implementujące jeden z opisanych poniżej interfejsów, odpowiadających czterem ich kategoriom, oraz interfejs `ConstructorInfo` umożliwiający pobieranie informacji o klasach i tworzenie interfejsu użytkownika pozwalającego na ich konfigurację.

5.5.2.1. Interfejs ConstructorInfo

Bazę systemu wtyczek stanowi interfejs `ConstructorInfo`. Jest to interfejs znacznikowy, tzn. nie zawierający żadnych metod, a używany jedynie do grupowania klas spełniających pewne wymagania. W interfejsie zdefiniowana jest klasa zagnieżdżona `Info` wykorzystywana do opisywania klas oraz parametrów ich konstruktorów. Zawiera ona jedynie pola publiczne, przechowujące krótki opis klasy oraz nazwy, opisy i typy parametrów przyjmowanych przez wybrany konstruktor klasy.

Klasa implementująca interfejs `ConstructorInfo` powinna zawierać statyczne pole o nazwie `info` typu `ConstructorInfo.Info`, wypełnione danymi tej klasy. Po-

winna ona także posiadać publiczny konstruktor przyjmujący jako jedyny parametr tablicę łańcuchów typu `String`. Powinien on interpretować kolejne elementy tej tablicy w sposób zgodny z opisami i typami zawartymi w polu `info`. Jako konwencję przyjmuje się, że podanie łańcucha pustego jako wartości pewnego parametru powinno oznaczać przyjęcie dla niego wartości domyślnej. Odniesienia do „parametrów konstruktora" w opisach poszczególnych klas w podpunktach punktu 5.6 dotyczą tego właśnie konstruktora.

Informacje zawarte w polu `info` są wykorzystywane w interfejsie użytkownika przez klasę `ClassConstructorPanel`, która na ich podstawie automatycznie tworzy opisane pola tekstowe do wprowadzania parametrów. Z wykorzystaniem wprowadzonych w nich danych i przy użyciu wspomnianego wyżej konstruktora przyjmującego jako parametr tablicę łańcuchów może później zostać stworzona instancja klasy.

5.5.2.2. Interfejs Filter

Interfejs `Filter` implementowany jest przez klasy realizujące różne funkcje wstępnego przetwarzania dokumentów tekstowych. Jedyną metodą interfejsu jest metoda `void filter(TextDocument doc)` powodująca przetworzenie dokumentu `doc`. Filtry działając na dokument mogą zmieniać zawarte w nim słowa, przy czym pozostawiają pole `word` bez zmian a modyfikują jedynie pole `lemma`. Mogą one także usuwać słowa z dokumentu, modyfikować ich wagi oraz zmieniać wszelkie inne cechy dokumentu, np. jego język.

5.5.2.3. Interfejs CoordinateGenerator

Klasy implementujące interfejs `CoordinateGenerator` służą do generowania zbiorów wektorów reprezentujących zbiory dokumentów tekstowych. Najważniejszą metodą jest `Map toCoords(Iterator docs,Filter[] filters)` powodująca przetworzenie zbioru dokumentów, którego przeglądanie umożliwia iterator `docs`.

Parametr `filters` stanowi tablicę filtrów, jakie w ramach wstępnego przetwarzania powinny zostać zastosowane na każdym z dokumentów. Uruchomienie filtrów dopiero w tej metodzie ma na celu zmniejszenie zużycia pamięci przez program. Dzięki temu, że wstępne przetwarzanie jest wykonywane dopiero tuż przed utworzeniem wektorowej reprezentacji dokumentu, znika konieczność przechowywania przetworzonych dokumentów w pamięci przez długi czas.

Zwykle jako iteratora `docs` używa się obiektów specjalnej klasy `DocumentsFromFilesIterator`. Iterator tej klasy tworzy odpowiednie obiekty typu `TextDocument` na podstawie zawartości plików dyskowych dopiero w momencie, gdy użytkownik żąda referencji do nich za pomocą metody `next()`. W typowych implementacjach metody `toCoords()` tak utworzony dokument jest poddawany działaniu filtrów, następnie jest przeglądany i w zmiennych lokalnych zapisywane są pewne informacje statystyczne, po czym obiekt może zostać zwolniony i ładowany jest następny dokument z dysku. W ten sposób dokumenty przechowywane są w pamięci jedynie przez krótki okres czasu.

Interfejs zawiera również metody: `featureNames()`, zwracającą listę opisów cech stanowiących bazę wykorzystywanej przestrzeni cech oraz `filteringTime()` zwracającą łączny czas wykonania wszystkich użytych filtrów w milisekundach.

5.5.2.4. Interfejs StatCollector

Zbieranie różnego rodzaju statystyk z dokumentów (np. zliczanie częstości słów), stanowiące często jeden z etapów tworzenia reprezentacji dokumentów w przestrzeniach wektorowych, jest zwykle wykonywane nie bezpośrednio w metodzie `toCoords()` klas implementujących interfejs `CoordinateGenerator`, lecz w specjalnych klasach implementujących interfejs `StatCollector`. Interfejs ten zawiera tylko jedną metodę, `Map getStats(TextDocument doc)`. Znaczenie zwracanej mapy zależne jest od rodzaju zbieranych statystyk.

5.5.2.5. Interfejs Metric

Klasy implementujące ten interfejs reprezentują w programie metryki przestrzeni liniowych i umożliwiają obliczanie odległości pomiędzy punktami tych przestrzeni. Jedyną metodą interfejsu jest metoda `double distance(Vector v1,Vector v2)` obliczająca odległość pomiędzy dwoma wektorami.

5.5.2.6. Interfejs Clusterizer

Interfejs `Clusterizer` implementowany jest przez klasy realizujące zadanie grupowania wektorów. Definiuje on metody pozwalające na przypisanie klasteryzatorowi zbioru wektorów do grupowania, wybór metryki (obiektu typu zgodnego z `Metric`), w której grupowanie ma przebiegać oraz na uruchomienie obliczeń i odczyt wyników.

5.6. Zaimplementowane algorytmy

5.6.1. Filtry

Do wstępnego przetwarzania tekstu służą filtry – klasy zgrupowane w pakiecie `edu.mgr.filter` implementujące interfejs `Filter` oraz interfejs `ConstructorInfo`. Wszystkie filtry operujące na lematach słów (z wyjątkiem `ToLowercase`) zakładają, że lematy słów wchodzących w skład dokumentu zostały już sprowadzone do samych małych liter.

5.6.1.1. DetectLanguage

`DetectLanguage` to filtr rozpoznający język dokumentu za pomocą stop-listy. Aby dać priorytet informacji o języku zawartej w metadanych dokumentu, nie robi on nic, jeżeli język dokumentu jest już znany. W sytuacji, gdy język nie jest jeszcze znany, klasa wykorzystuje załadowane do pamięci stop-listy (patrz opis klasy `StopList` w punkcie 5.6.1.6) i porównuje liczby wystąpień w tekście słów z poszczególnych list. Kryterium wyboru języka dokumentu stanowi występowanie w tekście stop-słów pewnego języka większą liczbę razy niż stop-słów pozostałych języków. W przypadku, gdy rozpoznanie języka nie powiodło się (np. dokument nie zawiera żadnego słowa ze stop-listy któregokolwiek języka), pole określające język dokumentu pozostawiane jest w stanie oznaczającym język nieznany.

Parametr konstruktora pozwala zdefiniować inny od domyślnego kod języka przypisywany dokumentom, dla których wykrycie języka nie było możliwe.

5.6.1.2. EgothorStemmer

Filtr `EgothorStemmer` zamienia lemat każdego słowa w dokumencie na wynik działania na niego stemmera Egothor. Wykorzystano oryginalną implementację autorów tego projektu oraz zestawy danych dla różnych języków pochodzące z projektów Stempel (język polski) oraz Egothor (pozostałe języki). Spośród dostępnych w bezpłatnej wersji projektu Stempel tablic dla języka polskiego wybrano zbiór `stemmer_2000.out`, wygenerowany na podstawie największego zbioru uczącego.

Jako rozszerzenie dodano możliwość pracy algorytmu w trybie rekurencyjnym. Przeprowadzone dla języka polskiego testy (patrz punkt 6.3.2.2.2) wskazują na to, że w tym trybie co prawda rdzenie produkowane przez stemmer znacznie częściej są niepodobne do lematów słów, natomiast rzadziej zdarza się generowanie różnych rdzeni

dla różnych form gramatycznych tego samego słowa. Na przykład słowo *obłędu* jest w algorytmie nierekurencyjnym sprowadzane do postaci *obłęd*, ale samo słowo *obłęd* przekształcane jest w *obłęda*. Użycie algorytmu rekurencyjnego pozwala sprowadzić obie formy do tej samej postaci. Niestety, w odróżnieniu od prostych stemmerów obcinających końcówki, jak np. stemmer Lovins, stemmer Egothor nie gwarantuje skracania rdzenia w każdej iteracji, a co za tym idzie, istnieje możliwość powstania nieskończonej pętli jeśli jedynym warunkiem stopu jest brak zmian rdzenia podczas dwóch kolejnych iteracji. Dlatego też w stworzonej implementacji wprowadzono sztywny limit maksymalnie dziesięciu iteracji stemmera dla każdego słowa.

Aby zaoszczędzić czas związany z ładowaniem z dysku danych stemmera, wprowadzono mechanizm pozwalający na wielokrotne wykorzystanie tych samych obiektów typu `EgothorStemmer`. W klasie zdefiniowano statyczną i finalną mapę, która łańcuchom stanowiącym kody języków przypisuje obiekty stemmera dopasowane do tych języków. Struktura ta jest inicjalizowana przy tworzeniu pierwszej instancji klasy, zaś późniejsze odwołania korzystają z załadowanych już do pamięci struktur danych. W sytuacji, gdy język przetwarzanego dokumentu nie jest obsługiwany ze względu na brak odpowiedniego pliku reguł, stemmer nie dokonuje żadnych zmian w dokumencie.

Konstruktor klasy pozwala podać nazwę pliku zawierającego reguły wykorzystywane przez stemmer. W przypadku takiej konstrukcji obiektu, zamiast z informacji o języku zawartej w obiekcie dokumentu i załadowanych już do pamięci zestawów reguł, instancja korzysta ze swojego własnego ich zestawu, ładowanego z pliku. W katalogu `data/EgothorStemmer` umieszczono pliki reguł dla języków: polskiego, angielskiego i niemieckiego, a dodanie dalszych języków jest możliwe przez pobranie z internetowej strony projektu Egothor i skompilowanie dodatkowych plików reguł. Konstruktor pozwala także wybrać rekurencyjny lub nierekurencyjny tryb pracy stemmera, przy czym dla każdej instancji klasy, nawet korzystającej z załadowanych uprzednio do pamięci danych, możliwy jest niezależny wybór trybu pracy.

5.6.1.3. FixApostrophes

`FixApostrophes` to prosty filtr, którego zadaniem jest wykrycie apostrofów używanych jako cudzysłowy i usunięcie ich z tekstu. Cudzysłowy bywają używane zarówno jako części słów (np. w słowie *can't*) jak również zamiast cudzysłowów. Dlatego

też nie są one traktowane jak zwykłe separatory słów i muszą być obsłużone oddzielnie. Filtr usuwa ze wszystkich słów apostrofy występujące w nich jako pierwszy lub ostatni znak, natomiast pozostawia apostrofy znajdujące się w środku słowa. W pewnych sytuacjach apostrofy mogą występować na początku lub końcu słów jako znaki wchodzące w ich skład, a nie jako cudzysłowy (np. na początku słowa w pewnych dialektach j. niemieckiego lub na końcu dopełniacza liczby mnogiej w języku angielskim).

5.6.1.4. LovinsStemmer

Filtr `LovinsStemmer` zamienia lemat każdego słowa w dokumencie na wynik działania na niego stemmera Lovins. Zaimplementowana odmiana algorytmu Lovins poprawia błąd literowy w oryginalnej pracy zawarty w opisie 30. zasady transformacyjnej (końcówkę `ent` zamienia na `ens`, a nie `end`), wskazany w [Porter, The Lovins stemmer]. Lista końcówek i związanych z nimi warunków zaimplementowana została za pomocą tablic mieszających, co pozytywnie wpływa na szybkość działania algorytmu.

Konstruktor klasy pozwala wybrać rekurencyjny lub nierekurencyjny tryb pracy stemmera.

5.6.1.5. Soundex

Działanie filtru `Soundex` polega na zastąpieniu lematu każdego słowa przez jego kod Soundex lub SoundexPL. Implementowany wariant algorytmu Soundex różni się nieco od jego podstawowej wersji: pierwsze litery słów są przetwarzane w identyczny sposób jak pozostałe, zamieniono również kolejność drugiego i trzeciego kroku, tzn. najpierw usuwane są podwójne głoski, a dopiero potem głoski o kodzie 0. Apostrof, pojawiający się w słowach typu *can't*, został dodany do grupy numer 0. Wprowadzono także dodatkową grupę (kod 9) dla znaków nie występujących w alfabecie angielskim.

Działanie algorytmu SoundexPL jest zgodne z jego opisem zawartym w punkcie 2.5.4.2, z tym, że znaki nie występujące w języku polskim są przypisywane do grupy numer 9.

Wybór wersji Soundex lub SoundexPL następuje poprzez podanie odpowiednio wartości `false` lub `true` jako parametru konstruktora klasy.

5.6.1.6. StopList

Klasa `StopList` implementuje algorytm stop-listy – filtr ten usuwa z dokumentu wszystkie wystąpienia słów zawartych w liście wczytywanej z pliku tekstowego. Plik taki korzysta z kodowania UTF-8 i zawiera po jednym słowie ze stop-listy w wierszu, z tym, że wiersze puste oraz rozpoczynające się od znaku `#` są ignorowane. Nie są rozróżniane małe i wielkie litery. W pliku dopuszczalne są powtórzenia. Katalog `data/StopList` zawiera listy dla języków: polskiego, angielskiego, niemieckiego i łacińskiego, pochodzące z różnych źródeł (projekt WordNet, projekt Snowball, wersja demo korpusu PWN, Perseus Digital Library[3]) i krytycznie poprawione oraz rozszerzone przez autora.

Za pomocą stop-listy możliwe jest także rozpoznawanie języka dokumentu. Sam algorytm rozpoznawania zaimplementowany został w klasie `DetectLanguage` (5.6.1.1), ale to klasa `StopList` dostarcza informacji o liczbie wystąpień słów z poszczególnych list w dokumencie.

Zarządzanie pamięcią, metody konstrukcji klasy oraz sposób wykorzystania zawartych w obiekcie dokumentu informacji o języku zorganizowane są podobnie jak w klasie `EgothorStemmer` (szczegółowy opis w punkcie 5.6.1.2).

5.6.1.7. StripUmlauts

`StripUmlauts` to trywialny filtr, którego celem jest dostosowanie dokumentu do sposobu działania niemieckojęzycznej wersji stemmera Egothor (5.6.1.2), która wymaga, by zastąpić niemieckie znaki diakrytyczne *ä*, *ö*, *ü*, *ß* ich odpowiednikami *a*, *o*, *u*, *ss*.

5.6.1.8. Thesaurus

Celem filtru `Thesaurus` jest poprawienie jakości grupowania dokumentów poprzez zastąpienie wszystkich wystąpień słów należących do pewnego zbioru wyrazów bliskoznacznych wystąpieniami pojedynczego słowa stanowiącego reprezentację tego zbioru.

Klasa odczytuje zawartość słownika z pliku o formacie wykorzystywanym przez edytor tekstu KWord, z tą różnicą, że plik powinien korzystać z kodowania znaków UTF-8, a nie z domyślnego kodowania znaków ustalonego w systemie operacyjnym.

3 http://www.perseus.tufts.edu/

Plik składa się z pewnej liczby wierszy. Wiersze zaczynające się od znaku `#` lub od dwóch spacji stanowią komentarz i są ignorowane. Każdy z pozostałych wierszy definiuje jedną grupę wyrazów bliskoznacznych. Wiersz taki składa się z dwóch części oddzielonych od siebie znakiem `#`. Każda część stanowi listę oddzielonych od siebie średnikami wyrażeń, które mogą zawierać spacje; bieżąca implementacja klasy `Thesaurus` ignoruje jednak wyrażenia składające się z więcej niż jednego wyrazu. Druga lista może być pusta, natomiast pierwsza musi zawierać przynajmniej jedno wyrażenie. Pierwsze wyrażenie pierwszej listy stanowi hasło, którego dotyczy dany wiersz, zaś pozostałe jej elementy stanowią synonimy tego hasła. Druga lista obejmuje słowa o znaczeniu szerszym niż hasło (ang. *hypernyms*). Nie są rozróżniane małe i wielkie litery. Przykładowy wiersz składający się z dwóch części wygląda następująco:

```
;car;auto;automobile;machine;motorcar;#;motor vehicle;automotive vehicle;
```

Zarządzanie pamięcią, metody konstrukcji klasy oraz sposób wykorzystania zawartych w obiekcie dokumentu informacji o języku zorganizowane są podobnie jak w klasie `EgothorStemmer` (patrz opis w punkcie 5.6.1.2).

Oprócz wyboru nazwy pliku zawierającego treść słownika (gdy nie korzystamy ze słowników załadowanych już do pamięci), konstruktor pozwala na ustalenie, czy słowa zawarte w drugiej części każdego wiersza powinny być traktowane na równi z synonimami, czy też powinny być ignorowane. Tę ostatnią własność można zmienić również po konstrukcji obiektu.

Oprócz typowego zastosowania polegającego na sprowadzaniu do wspólnej postaci wyrazów bliskoznacznych, klasa może być wykorzystana jako prosta forma słownikowego lematyzatora. W zasadzie wykorzystywane struktury danych nie są optymalne dla przechowywania typowych dla lematyzatorów słowników liczących setki tysięcy pozycji. Pomimo to, użycie nawet uproszczonego słownika, np. zawierającego tylko słowa o nieregularnej odmianie, których obsługa przez stemmer algorytmiczny jest niepoprawna, może znacznie poprawić jakość działania całego procesu przetwarzania tekstu.

Ponieważ każde słowo (w sensie ciągu znaków) może posiadać wiele różnych znaczeń (homonimów), może istnieć kilka niezależnych grup synonimów, z których każda zawiera to słowo, ale w innym jego znaczeniu. Przykładowo, w słowniku mogłyby się równocześnie znaleźć grupy synonimów (*zamek*, *forteca*, *twierdza*) i (*za-*

mek, *blokada drzwi*). Bez przeprowadzenia dogłębnej analizy tekstu stwierdzenie, czy pewne wystąpienie słowa *zamek* w tekście należy interpretować jako należące do pierwszej czy do drugiej grupy, nie jest możliwe. Dlatego też klasa `Thesaurus` podczas budowy swoich wewnętrznych struktur danych na podstawie struktury słownika zapisanej w pliku przyjmuje założenie, że sytuacja taka, jak opisana powyżej nigdy nie ma miejsca. Jeżeli więcej niż jedna grupa synonimów zawiera to samo słowo, słowa z obu grup mogą, ale nie muszą zostać zaliczone do tej samej grupy i być zastąpione w tekście przez to samo słowo ją reprezentujące.

Wykorzystywane w praktyce słowniki wyrazów bliskoznacznych, np. współpracujące z edytorami tekstu, zwykle zawierają homonimy należące do różnych grup synonimów. Większość języków naturalnych posiada znaczną liczbę homonimów, a najpopularniejszy w internecie język angielski ma ich szczególnie wiele. Można zatem przy korzystaniu z klasy `Thesaurus` wybrać jedną z dwóch opcji: albo korzystać z gotowych słowników, co wiąże się z potencjalnie przypadkowym uznawaniem synonimów słów stanowiących homonimy za wzajemne synonimy lub nie, albo przygotować odpowiednio sformatowane słowniki, w których grupy synonimów zostały dobrane tak, aby uniknąć tego rodzaju wątpliwości.

Ze względu na łatwą dostępność w internecie gotowych słowników wyrazów bliskoznacznych, wybrano pierwsze z przedstawionych rozwiązań, choć oczywiście nic nie stoi na przeszkodzie, by również utworzyć w osobnych plikach uproszczone słowniki nie zawierające homonimów. W projekcie zawarto słowniki wyrazów bliskoznacznych dla języków angielskiego oraz polskiego wykorzystywane przez edytor tekstu KWord.

5.6.1.9. ToLowercase

`ToLowercase` to filtr zamieniający wszystkie lematy słów wchodzących w skład dokumentu na same małe litery. Zamiana jest wykonywana zgodnie z ustawieniami językowymi systemu operacyjnego (w przypadku systemów uniksowych ustawienia te są określone przez zmienne środowiskowe `LC_CTYPE` i `LC_ALL`). Filtr ten powinien zwykle być pierwszym filtrem stosowanym na każdym dokumencie, gdyż do poprawnego działania większości pozostałych filtrów konieczne jest, by słowa składały się wyłącznie z małych liter.

5.6.1.10. WeighPartsOfSpeech

Filtr ten stanowi prosty przykład wykorzystania informacji o budowie gramatycznej dokumentu. Zmienia on wagi słów mnożąc je przez czynniki zależne od tego, jaką częścią mowy jest dane słowo. Waga spójników zmniejsza się, natomiast waga rzeczowników i czasowników mnożona jest przez liczbę większą od jedności celem podkreślenia ich znaczenia w dokumencie. Jeśli część mowy nie jest określona (wartość `CM_INNA` pola `czescMowy`), waga słowa nie ulega zmianie. W szczególności, taka sytuacja ma miejsce gdy dokument został wczytany ze zwykłego pliku tekstowego i informacje o gramatyce nie są w ogóle dostępne.

5.6.2. Generatory współrzędnych

Generatory współrzędnych (konwertery) to klasy implementujące interfejsy `CoordinateGenerator` i `ConstructorInfo`, używane do zamiany zbiorów dokumentów tekstowych przechowywanych w obiektach typu `TextDocument` na ich reprezentacje w postaci wektorów. Zostały one zgrupowane w pakiecie `edu.mgr.converter`.

5.6.2.1. Null

`Null` to generator współrzędnych, który dla każdego dokumentu generuje wektor zerowy jako odpowiadający mu wektor. Celem użycia klasy może być skrócenie oczekiwania na wyniki w testach mierzących jedynie czas wykonania filtrów.

5.6.2.2. TopFrequency i RAMTopFrequency

Klasa `TopFrequency` bada częstość występowania w dokumentach poszczególnych n-gramów i jako współrzędne dokumentów zwraca częstości względne występowania w nich pewnej ustalonej liczby n-gramów o największej sumie częstości względnych we wszystkich dokumentach.

Wartość n (rozmiar n-gramów), to czy mają być rozpatrywane n-gramy uporządkowane czy nie, oraz długość generowanych wektorów mogą być ustalone przy konstrukcji obiektu. Jako parametr konstruktora można także podać sposób obliczania częstości: albo względem liczby wszystkich występujących n-gramów, albo względem liczby różnych n-gramów.

Klasa `RAMTopFrequency` jest niemal identyczna z klasą `TopFrequency` poza tym, że przechowuje wszystkie dane w pamięci operacyjnej, w związku z czym działa szyb-

ciej dla małych zbiorów danych, natomiast szybciej też wyczerpuje dostępną pamięć. Dodatkowo umożliwia ona również rezygnację z redukcji wymiaru przestrzeni cech i znalezienie wektorów w przestrzeni o niezredukowanym wymiarze. W tym celu należy parametrowi `length` nadać wartość 0.

5.6.2.3. TopSAI i RAMTopSAI

Konwerter `TopSAI` wiąże współrzędne przestrzeni, w której reprezentowane są dokumenty z n-gramami o największym współczynniku SAI (opisanym w punkcie 3.3.3). Wartość poszczególnych współrzędnych dla konkretnego dokumentu jest obliczana jako współczynnik TFIDF odpowiadających tym współrzędnym n-gramów.

Konstruktor klasy pozwala na podanie maksymalnej długości generowanych wektorów, rozmiaru n-gramów oraz czy mają być zliczane n-gramy uporządkowane, czy nie. Parametr `useGF` określa, czy ma być stosowana rozszerzona wersja algorytmu SAI, czy wersja podstawowa.

Klasa `RAMTopSAI` jest niemal identyczna z klasą `TopSAI` poza tym, że przechowuje wszystkie dane w pamięci operacyjnej, w związku z czym działa szybciej dla małych zbiorów danych, natomiast szybciej też wyczerpuje dostępną pamięć. Dodatkowo umożliwia ona również rezygnację z używania algorytmu SAI i znalezienie wektorów w przestrzeni o niezredukowanym wymiarze. W tym celu należy parametrowi `length` nadać wartość 0.

5.6.3. Metryki

Klasy reprezentujące metryki znajdują się w pakiecie `edu.mgr.clusterizer` i implementują interfejsy `Metric` oraz `ConstructorInfo`. Dostępne są następujące klasy metryk:

- metryka kosinusowa (klasa `Cosine`)
- metryka euklidesowa (klasa `Euclidean`)
- „metryka” Jacquarda (klasa `Jacquard`)
- metryka Manhattan (klasa `Manhattan`)
- metryka maksimum (klasa `Max`)
- metryka wzorowana na metryce Hamminga (klasa `QuasiHamming`).

5.6.4. Klasteryzatory

Klasteryzatory zgrupowane zostały w pakiecie `edu.mgr.clusterizer`. Są to implementujące interfejsy `Clusterizer` oraz `ConstructorInfo` klasy służące do grupowania wektorów reprezentujących dokumenty tekstowe.

5.6.4.1. HwhKMeans

Klasa `HwhKMeans` implementuje dwa różne algorytmy grupowania: metodę HWH (patrz punkt 4.3.2) oraz zmodyfikowaną metodę k-means (patrz punkt 4.3.1). Autorem klasy jest Grzegorz Kolendo. Klasa stanowi jedną z pierwszych wersji opracowanego przez niego klasteryzatora, odmienną od opisanej w ostatecznej redakcji jego pracy magisterskiej.

Pierwszy z dwóch parametrów konstruktora pozwala wskazać, który ze wspomnianych dwóch algorytmów ma zostać użyty (domyślnym jest HWH), zaś drugi określa liczbę iteracji algorytmu.

5.6.4.2. Null

Klasteryzator `Null` nie wykonuje żadnych obliczeń – jako wynik działania zwraca on zawsze pusty zbiór klastrów. Jest on wykorzystywany w celu skrócenia czasu oczekiwania na wyniki w testach mierzących jedynie czas wykonania filtrów.

5.6.5. Nadawanie klastrom nazw

Nadawanie klastrom nazw realizowane jest przez klasę `ClusterNamer`. Ponieważ zaimplementowano tylko jeden algorytm, nie został stworzony osobny interfejs. Nadawanie nazw odbywa się poprzez porównanie centroidu rozpatrywanego klastra z wektorem stanowiącym średnią arytmetyczną położeń wszystkich pozostałych znalezionych klastrów. Z wektora stanowiącego ich różnicę wybierana jest współrzędna o największej wartości i nazwa cechy jej odpowiadającej jest zwracana jako nazwa klastra. Klasa zawiera dwie metody, z których jedna służy do znajdowania nazwy pojedynczego klastra, zaś druga pozwala na znalezienie nazw wszystkich klastrów w pewnym zbiorze w sposób bardziej wydajny niż wielokrotne wywoływanie pierwszej.

5.7. Instrukcja obsługi

5.7.1. Kompilacja

Projekt jest automatycznie kompilowany po otwarciu go w środowisku Eclipse. Możliwa jest także kompilacja w trybie wsadowym za pomocą narzędzia Ant (należy wydać polecenie `ant` w głównym katalogu projektu). Plik projektu dla Anta (`build.xml`) został wygenerowany automatycznie za pomocą wtyczki eclipse2ant[4] z odpowiednich plików używanych przez Eclipse i musi być odtwarzany za każdym razem, gdy wprowadzane są zmiany w opcjach kompilacji projektu w Eclipse.

5.7.2. Uruchamianie

Uruchomienie programu następuje poprzez wywołanie metody `main()` zawartej w klasie `GUI`, co może nastąpić w wyniku wykonania polecenia `java edu.mgr.GUI` w głównym katalogu programu lub przez uruchomienie skryptu startowego (`run` dla Linux/Unix, `run.bat` dla Windows).

Uruchomienie programu w trybie interaktywnym następuje w wyniku uruchomienia go bez żadnych parametrów. Jeżeli jako parametr programu podana zostanie opcja `--run N`, program uruchomiony zostanie w trybie wsadowym. W trybie tym nie jest wyświetlany interfejs użytkownika, a program automatycznie wykonuje `N` razy obliczenia dla parametrów zapisanych w pliku konfiguracyjnym, po czym kończy pracę. W przypadku niepodania wartości `N`, obliczenia wykonywane są jeden raz.

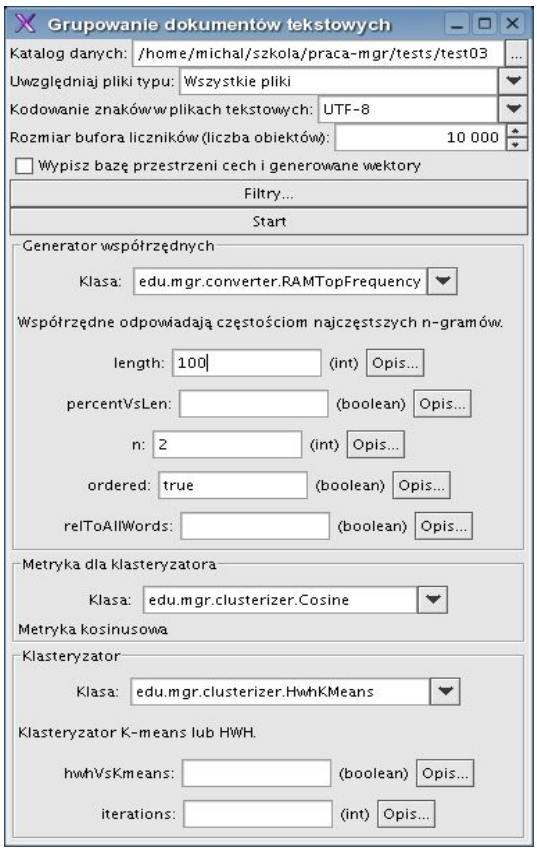
Również niektóre inne klasy posiadają metody `main()`, które służą do testowania poprawności ich działania. W większości przypadków pozwalają one na podanie danych testowych na standardowym wejściu i wypisują wynik działania implementowanego algorytmu na standardowym wyjściu.

5.7.3. Korzystanie z programu

W górnej części okna programu znajdują się pola pozwalające wybrać katalog z danymi wejściowymi, typ plików, które mają zostać zgrupowane (wszystkie pliki, pliki tekstowe, pliki HTML lub pliki XML) oraz kodowanie znaków używane przy odczycie plików tekstowych. Kodowanie dla plików HTML i XML jest ustalane na podstawie odpowiednich znaczników w ich treści, a w razie braku tych znaczników używane jest domyślne kodowanie znaków ustalone w systemie operacyjnym. Pole *Rozmiar bufora*

4 http://www.geocities.com/richard_hoefter/eclipse2ant/

liczników pozwala określić maksymalną liczbę obiektów przechowywanych równocześnie w pamięci przez obiekty typu `BufferedCounterMap` (patrz punkt 5.4). Włączenie opcji *Wypisz bazę przestrzeni cech i generowane wektory* powoduje zamieszczenie odpowiednich informacji w wynikach działania programu. Jeżeli nie jest stosowana redukcja wymiaru przestrzeni cech (parametr *length* wybranego generatora współrzędnych ma wartość 0), objętość wyników może w wyniku użycia tej opcji wzrosnąć o wiele megabajtów.

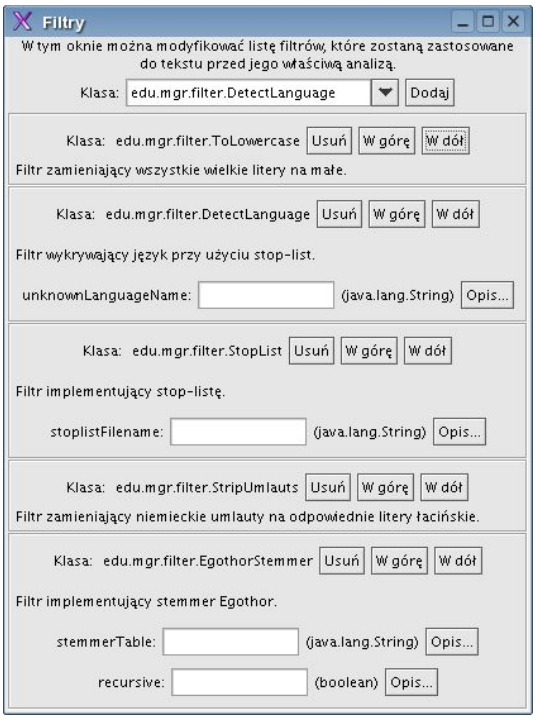


Poniżej znajdują się przyciski: *Filtry...* otwierający okno wyboru filtrów oraz *Start* uruchamiający obliczenia.

W głównej części okna znajdują się panele pozwalające na wybór klas realizujących zadania generacji współrzędnych i klasteryzacji oraz klasy metryki używanej przez klasteryzator. Wybór klasy do każdego z tych celów następuje przez wybranie odpowiedniej nazwy klasy z rozwijanej listy. W wyniku wybrania w liście pewnej klasy, wygląd panelu zmienia się tak, by umożliwić wpisanie parametrów przyjmowanych przez konstruktor tej klasy i wpływających na sposób działania związanego z nią algorytmu. Dla każdego parametru jest wyświetlana jego nazwa i typ, a po kliknięciu na przycisk *Opis...* pojawia się okno z krótkim opisem wyjaśniającym jego znaczenie. Pozostawienie pewnych pól pustych powoduje przyjęcie dla odpowiednich parametrów wartości domyślnych.

Okno wyboru filtrów posiada rozwijaną listę wszystkich dostępnych klas filtrów. Wybór pewnej klasy a następnie kliknięcie przycisku *Dodaj* powoduje dodanie do okna nowego panelu wyboru parametrów klasy (podobnego do opisanych w poprzednim paragrafie). Panele te posiadają przyciski pozwalające na przesuwanie ich względem siebie w górę i w dół oraz na usuwanie ich z okna. Ustalona w oknie kolej-

ność paneli od góry ku dołowi oznacza kolejność, w jakiej filtry będą stosowane do analizowanych dokumentów.



Aby uruchomić proces grupowania dokumentów, należy kliknąć przycisk *Start*. Wyniki oraz informacje o ewentualnych błędach są wypisywane odpowiednio na standardowym wyjściu i standardowym strumieniu błędu, co umożliwia proste ich przekierowanie do pliku lub na wejście innego programu za pomocą standardowych mechanizmów powłoki.

Aplikacja sygnalizuje fakt wykonywania obliczeń przez zmianę kursora myszy. Obliczenia wykonywane są w głównym wątku aplikacji, bez wydzielania osobnych wątków roboczych i wątków interfejsu użytkownika. Skutkuje to brakiem odświeżania GUI podczas trwania obliczeń.

5.7.4. Modyfikacja pliku konfiguracyjnego

Stan interfejsu użytkownika jest zapisywany do pliku konfiguracyjnego opisanego w punkcie 5.2. Ręczna modyfikacja tego pliku nie jest nigdy konieczna podczas normalnego użytkowania programu, ale może się okazać niezbędna jeśli podczas modyfikacji programu nastąpi zmiana katalogów, w których są przechowywane pliki klas wtyczek, lub jeżeli ten sam plik konfiguracyjny jest wykorzystywany podczas uruchamiania programu w systemach operacyjnych stosujących różne znaki do oddzielania elementów ścieżki pliku (np. Linux i Windows). W takim wypadku aplikacja po odczytaniu niepoprawnych danych konfiguracyjnych nie jest w stanie poprawnie utworzyć paneli. Należy wtedy za pomocą edytora tekstowego usunąć z pliku konfiguracyjnego wiersze przechowujące konfigurację paneli (każdy wiersz zawierający klucz zaczynający się od `panel` oraz dwa następujące po nim) i ponownie uruchomić program. Zostaną wtedy utworzone panele znajdujące się w domyślnym stanie.

5.7.5. Skrypty pomocnicze

W tabeli 4 przedstawiono listę skryptów wykorzystywanych do tworzenia i analizy danych testowych. Skrypty te znajdują się w głównym katalogu aplikacji.

Nazwa	***Opis***
`file-stats`	Skrypt wypisuje liczbę plików znajdujących się w bieżącym katalogu, ich średni rozmiar oraz odchylenie standardowe rozmiaru. Podkatalogi są pomijane.

`generate-text-samples`	Skrypt przyjmuje następujące parametry (opcjonalne w nawiasach kwadratowych):
	`plik_wejsciowy.rozsz`
	`[liczba_fragmentow]`
	`[rozmiar_fragmentow]`
	Z pliku `plik_wejsciowy.rozsz` wczytywany jest tekst, z którego jest wybieranych losowo `liczba_fragmentów` fragmentów o długości równej w przybliżeniu `rozmiar_fragmentów`. Granice fragmentów są wybierane tak, by znalazły się w nich tylko całe słowa. Znalezione fragmenty zapisywane są do plików o nazwach `plik_wejsciowy-NNNN.rozsz` w bieżącym katalogu, gdzie `NNNN` oznacza numer kolejny.

`list-nontrivial-clusters`	Traktując standardowe wejście jako wynik działania głównej aplikacji, skrypt wypisuje na standardowym wyjściu znalezione klastry składające się z więcej niż jednego dokumentu.
`remove-vectors-and-base`	Traktując standardowe wejście jako wynik działania głównej aplikacji, skrypt usuwa z niego listę wektorów bazy przestrzeni cech oraz listę wektorów wygenerowanych dla poszczególnych dokumentów. Wynik wypisywany na standardowym wyjściu jest sformatowany tak samo, jak gdyby główna aplikacja została od razu uruchomiona z wyłączoną opcją wypisywania bazy i wektorów. Skrypt zakłada, że systemowym kodowaniem znaków jest UTF-8.

Tabela 4: Skrypty pomocnicze

6. Testy

W niniejszym rozdziale omówione zostały testy programu przeprowadzone w celu porównania wyników uzyskiwanych przy użyciu różnych algorytmów. Szczególny nacisk położono na zbadanie przydatności zaproponowanych w niniejszej pracy algorytmów SoundexPL oraz SAI.

6.1. Zestawy danych testowych

Jako zestawy danych testowych wykorzystane zostały głównie dokumenty lub ich fragmenty publicznie dostępne, znajdujące się w domenie publicznej lub podlegające otwartym licencjom dokumentacji. Wśród tekstów tych znalazły się przede wszystkim dzieła literatury światowej różnych okresów oraz dokumentacja techniczna. Wykorzystane zostały także przykładowe teksty polskich ustaw oraz stenogramy posiedzeń sejmowych komisji śledczych. Ponadto wykorzystano korpus *20 Newsgroups* składający się z 20000 artykułów pochodzących z 20 różnych internetowych grup dyskusyjnych. Użyto wersji oznaczonej kodem `20news-18828`, pozbawionej duplikatów wiadomości oraz nagłówków innych niż temat i adres nadawcy.

Fragmenty dokumentów generowane były w dwojaki sposób. Część z nich została wybrana przypadkowo, ale z uwzględnieniem granic paragrafów, tak aby w każdym fragmencie znalazły się jedynie pełne paragrafy. Takie fragmenty zostały wydzielone z tekstów ręcznie. Inne fragmenty zostały wygenerowane automatycznie za pomocą skryptu `generate-text-samples` (patrz opis w punkcie 5.7.5). Do testów użyto zarówno zbiorów dokumentów o zbliżonej długości jak i zbiorów zawierających dokumenty o zróżnicowanych rozmiarach. Wszystkie przygotowane do testów dokumenty i ich fragmenty zostały przekonwertowane do kodowania znaków UTF-8. Dokumenty pobrane ze strony projektu Gutenberg[5] pozbawiono znajdujących się na początku i na końcu informacji o tym projekcie, mogących wpłynąć na proces grupowania.

W tabeli 5 przedstawiono charakterystykę poszczególnych zestawów danych testowych. Dokumenty wchodzące w skład każdego z nich znajdują się w podkatalogu `tests/<nazwa zestawu>`. Statystyki dotyczące rozmiarów plików obliczono za pomocą skryptu `file-stats`.

5 http://promo.net/pg/

Nazwa	*Liczba plików*	*Średni rozmiar [kB]*	*Odch. std. rozmiaru [kB]*	*Języki*	*Opis*
test00	6	0,49	0,29	PL	Nonsensowne teksty na temat zdania *Ala ma kota*. Zestaw służy do testowania poprawności wczytywania dokumentów w formatach HTML i XML, a nie do testowania algorytmów.
test01	26	20,0	39,1	EN, PL	6 dużych (od kilkunastu do ponad stu kilobajtów) dokumentów technicznych typu HOWTO oraz krótkie (kilka kB) fragmenty utworów literackich.
test02	9	4,65	1,44	EN, LA	Fragmenty tekstów literackich i dokumentów prawnych w j. angielskim oraz *Wojny galijskiej* Juliusza Cezara.
test03	14	4,68	1,99	DE, LA	Fragmenty utworów literackich.
test04	10	3,54	1,37	PL	Fragmenty utworów literackich.
test05	56	50,7	65,0	EN	Dokumenty typu HOWTO.
test06	12	50,8	47,3	EN	Dokumenty typu RFC.
test07	40	4,00	0,00	EN	Fragmenty utworów literackich.
test08	20	4,88	0,00	PL	Fragmenty *Wesela* Stanisława Wyspiańskiego.
test09	14	107	78,9	PL	Teksty prawne: ustawy i ordynacje wyborcze.
test10	38	12,9	7,24	EN	Fragmenty tekstów literackich oraz dokumentów prawnych.
test11	8	232	211	EN	Wybór tekstów literackich oraz dokumentów prawnych.
test12	9	299	275	EN	Wybór tekstów literackich oraz dokumentów prawnych identyczny z test11 z dodanym tekstem *Drakuli* Brama Stokera.
test13	990	1,27	1,41	EN	Korpus `20news-18828`, wiadomości z grupy `rec.autos`.
test14	3949	1,72	3,31	EN	Korpus `20news-18828`, wiadomości z grup `sci.*`.
test15[6]	18828	1,76	3,88	EN	Korpus `20news-18828`, całość.
test16	19	220	175	PL	Protokoły posiedzeń sejmowych komisji śledczych.

6 Nie udało się przeprowadzić żadnych testów dla tego zestawu danych. Z dwóch dostępnych maszyn testowych jedna dysponowała zbyt małą ilością pamięci operacyjnej i ze względu na ciągłe korzystanie z partycji wymiany obliczenia odbywały się w tak powolnym tempie, że ostatecznie zostały przerwane, zaś na drugiej każda próba uruchomienia programu z zestawem danych wejściowych `test15` kończyła się po kilku minutach krytycznym błędem JVM i prośbą o wysłanie zgłoszenia błędu do firmy Sun (zarówno dla JVM w wersji 1.4 jak i 1.5), pomimo że inne zestawy danych nie powodowały takich błędów.

Nazwa	*Liczba plików*	*Średni rozmiar [kB]*	*Odch. std. rozmiaru [kB]*	*Języki*	*Opis*
test17	552	0,254	0,153	EN	Opisy pakietów wchodzących w skład dystrybucji Slackware Linux 10.0. Z opisów usunięto znajdujące się na początku każdego wiersza nazwy pakietu.
test18	154	6,30	13,5	PL	Bajki Jana Brzechwy i Ignacego Krasickiego w formacie HTML.
test19	154	4,67	12,4	PL	Pliki z `test18` przekonwertowane do formatu tekstowego.

Tabela 5: Zbiory danych testowych

6.2. Nazewnictwo plików z wynikami testów

Wyniki wybranych testów wykonanych na poszczególnych zbiorach danych umieszczone zostały w katalogach `tests-results/<nazwa zestawu>`. Testy inne niż testy wydajności przeprowadzane były w zróżnicowanych warunkach, zatem zapisane w ich wynikach czasy wykonania nie są miarodajne.

Nazwa każdego z plików zawierających wyniki przedstawia opis użytych algorytmów. Nazwy plików zawierających wyniki odpowiednio wykonanych testów wydajności (patrz punkt 6.4.1) zaczynają się dla odróżnienia od pozostałych od przedrostka `perf_`. Właściwa nazwa występująca po ewentualnym przedrostku składa się z czterech oddzielonych od siebie znakami podkreślenia (_) części odpowiadających kolejno: użytemu zestawowi filtrów, generatorowi współrzędnych, metryce i klasteryzatorowi. Zestaw filtrów stanowi listę oddzielonych znakami minus (-) opisów klas, która może być pusta. Każda z pozostałych trzech części nazwy składa się z pojedynczego opisu klasy. Opis klasy składa się ze skrótu nazwy klasy, po którym może następować umieszczona w nawiasach okrągłych lista oddzielonych przecinkami trójek: nazwa parametru, znak `@`, wartość parametru. Parametry o wartościach domyślnych mogą zostać pominięte. Znaki separatorów zostały dobrane tak, by pliki wyników mogły być przechowywane na partycjach z systemem plików VFAT. Stosowane skróty nazw klas przedstawiono w tabeli 6. Nazwa pliku może kończyć się znakiem podkreślenia i liczbą pozwalającą odróżnić poszczególne testy wykonane z identycznymi parametrami.

Skrót	Klasa
Filtry	
detl	DetectLanguage
egothor	EgothorStemmer
fixa	FixApostrophes
lovins	LovinsStemmer
soundex	Soundex
stopl	StopList
stripu	StripUmlauts
thes	Thesaurus
lowc	ToLowercase
weigh	WeighPartsOfSpeech
Generatory współrzędnych	
null	Null
rtf	RAMTopFrequency
rtsai	RAMTopSAI
tf	TopFrequency
tsai	TopSAI
Metryki	
cos	Cosine
eucl	Euclidean
jacq	Jacquard
manh	Manhattan
max	Max
hamm	QuasiHamming
Klasteryzatory	
hwh	HwhKMeans
null	Null

Tabela 6: Skróty nazw klas

Przykład: `perf_lowc-detl-stopl-lovins_rtsai(length@100,n@2)_cos_hwh` oznacza test wydajności, w którym wykorzystano filtry: `ToLowercase`, `DetectLanguage`, `StopList`, `LovinsStemmer`, konwerter `RAMTopSAI` z parametrem `length` równym 100 i parametrem `n` równym 2, metrykę kosinusową oraz klasteryzator `HwhKMeans` z domyślnymi parametrami.

6.3. Subiektywna ocena wyników działania

Celem testów opisanych w tym punkcie było zbadanie wpływu wyboru różnych algorytmów i ich parametrów na poszczególnych etapach obliczeń na końcowy wynik działania programu.

6.3.1. Sposób testowania

Przy ocenie wyników działania brany był pod uwagę jedynie wynik końcowy, czyli zbiór znalezionych klastrów, gdyż ocena „poprawności” wygenerowanych wektorów w wielowymiarowej przestrzeni jest w większości wypadków niemożliwa do przeprowadzenia przez człowieka. Pewne ograniczenie metody polegającej na ocenie jedynie klastrów stanowi fakt, że zakłada ona całkowicie „poprawne” działanie klasteryzatora.

Fakt wykorzystania gotowej klasy klasteryzatora z jednej strony uprościł tworzenie aplikacji, ale z drugiej wpłynął negatywnie na jej elastyczność. Wszystkie wykorzysta-

ne algorytmy klasteryzacji to algorytmy dzielące, a więc w przypadku, gdy znajdowany zbiór klastrów odbiegał od oczekiwań, nie było możliwe prześledzenie drzewa klastrów w celu wykrycia przyczyny wykonania grupowania w dany sposób. Również brak możliwości zmiany domyślnej siły klasteryzacji w wykorzystanej implementacji klasteryzatorów należy uznać za jej poważną wadę. Wykorzystuje ona ponadto pewną liczbę stałych o wymiarze długości. Porównywanie odległości między wektorami z tymi stałymi może być zasadne w przypadku stosowania metryki kosinusowej lub innej metryki przyjmującej ograniczony od góry zbiór wartości. W przypadku metryk takich, jak euklidesowa, mogących osiągać dowolne nieujemne wartości, porównywanie odległości ze stałą niezależną od wymiaru i rozmiarów zbioru danych nie jest dobrą metodą sprawdzania przynależności do klastra. Wykorzystanie owych stałych skutkuje bardzo niewielką przydatnością obu klasteryzatorów do grupowania punktów w metrykach przyjmujących nieograniczone wartości oraz spadkiem siły grupowania przy wzroście wymiaru przestrzeni cech nawet w przypadku metryki kosinusowej. Problem próbowano obejść przez dopasowanie zbioru wektorów do wymagań klasteryzatora: normalizację wektorów w przypadku klasteryzatora HWH i przeskalowanie wszystkich wektorów tak, by mieściły się w hiperześcianie o boku 1 w przypadku zmodyfikowanego klasteryzatora k-means. Ponieważ jednak w przypadku np. metryki euklidesowej normalizacja wektorów bardzo silnie wpływa na odległości między nimi, omówione rozwiązanie nie było w stanie skutecznie przezwyciężyć wymienionych wad zaimplementowanych klasteryzatorów. Prawdopodobnie wykorzystanie ich ostatecznej wersji zamiast wersji roboczej pozwoli w przyszłości uniknąć większości wymienionych problemów.

Zarówno klasteryzator HWH jak i k-means przeprowadzają grupowanie w sposób niedeterministyczny, zależny od wybieranych losowo początkowych położeń centroidów klastrów. W większości przypadków różnice między poszczególnymi wykonaniami tego samego zadania były na tyle małe, że znajdowany był zawsze lub prawie zawsze ten sam zbiór klastrów. Ponieważ wykorzystywane liczby pseudolosowe są generowane za pomocą deterministycznego generatora, identyczne ciągi wyników kolejnych obliczeń otrzymuje się w każdym kolejnym uruchomieniu programu.

Oba klasteryzatory w wykorzystanej wersji testowej wykonują ustaloną z góry liczbę iteracji zamiast sprawdzania, czy między kolejnymi iteracjami dochodzi do

przenoszenia punktów z klastra do klastra. Taki sposób działania wpłynął zarówno na wyniki (brak dopasowania liczby iteracji do charakterystyki danych wejściowych), jak i na wydajność (wykonywanie w wielu sytuacjach zbyt dużej lub zbyt małej w stosunku do potrzeb liczby iteracji).

Z przedstawionych w dalszej części rozdziału przykładowych wyników działania programu usunięto klastry składające się z pojedynczych dokumentów. Wykorzystano w tym celu skrypt `list-nontrivial-clusters`. Wszystkie wyniki, w tym również te, których nie omawiano szczegółowo w pracy, znajdują się na płycie CD dołączonej do pracy. Ze względu na oszczędność miejsca nie zawierają one baz przestrzeni cech ani wektorów odpowiadających poszczególnym dokumentom (mogą one łatwo zostać wygenerowane w razie potrzeby).

6.3.2. Filtry

Testy pokazały, że stosowany zestaw filtrów ma duży wpływ na współrzędne generowanych wektorów, przy czym wpływ ten jest zwykle silniejszy w przypadku ważenia cech za pomocą częstości niż wtedy gdy używany jest TFIDF.

6.3.2.1. Zamiana na małe litery i stop-lista

Porównanie wyników grupowania przeprowadzonego dla tych samych danych wejściowych z użyciem podstawowych filtrów tekstu i bez nich prowadzi do wniosku, że grupowanie przeprowadzone na tekstach nie poddanych filtrowaniu często skutkuje umieszczeniem w jednej grupie dokumentów znacznie się różniących. Zjawisko to jest szczególnie widoczne w przypadku stosowania częstości jako wag. Użycie wag opartych na TFIDF łagodzi problem, gdyż w pewnym stopniu zastępuje wykorzystanie stop-listy: większość słów znajdujących się na stop-liście występuje w prawie wszystkich dokumentach (o ile są one dostatecznie długie i wszystkie napisane w jednym języku) i tym samym otrzymuje w niemal każdym tekście wartość TFIDF bliską zeru.

Przykładowy klaster znaleziony w teście `_rtf(length@0)_cos_hwh` dla danych `test01` jest typowy dla sytuacji nie stosowania stop-listy i wykorzystania częstości jako wag słów:

```
------ Klaster: UnorderedNGram[the]
heart_of_darkness-1.txt
Kernel-HOWTO
lord_jim-4.txt
lord_jim-1.txt
CrimeAndPunishment-3.txt
Apache-Overview-HOWTO
Unicode-HOWTO
```

```
heart_of_darkness-4.txt
CrimeAndPunishment-1.txt
Bzip2
CrimeAndPunishment-2.txt
lord_jim-2.txt
CrimeAndPunishment-4.txt
heart_of_darkness-3.txt
Apache-Compile-HOWTO
lord_jim-3.txt
Kernel-README
heart_of_darkness-2.txt
```

W klastrze tym znalazły się wszystkie dokumenty w języku angielskim należące do zbioru. Prawie wszystkie pozostałe dokumenty, napisane w języku polskim, również znalazły się w pojedynczym klastrze. Wykorzystanie TFIDF jako wag przy identycznych pozostałych parametrach doprowadziło do wykrycia następujących klastrów o więcej niż jednym elemencie:

```
------ Klaster: UnorderedNGram[kernel]
Kernel-HOWTO
Kernel-README

------ Klaster: UnorderedNGram[his]
lord_jim-4.txt
lord_jim-1.txt

------ Klaster: UnorderedNGram[Sonia]
CrimeAndPunishment-3.txt
CrimeAndPunishment-4.txt
```

```
------ Klaster: UnorderedNGram[Apache]
Apache-Overview-HOWTO
Unicode-HOWTO
Apache-Compile-HOWTO

------ Klaster: UnorderedNGram[Kurtz]
heart_of_darkness-4.txt
lord_jim-2.txt
```

Dla porównania, po zastosowaniu podstawowych filtrów (zamiana na małe litery, usunięcie apostrofów pełniących rolę cudzysłowów, stop-lista), wyniki otrzymane przy wykorzystaniu częstości jako wag przedstawiały się następująco:

```
------ Klaster: UnorderedNGram[kernel]
Kernel-HOWTO
Apache-Overview-HOWTO
Unicode-HOWTO
Bzip2
Apache-Compile-HOWTO
Kernel-README

------ Klaster: UnorderedNGram[sonia]
CrimeAndPunishment-3.txt
CrimeAndPunishment-2.txt
```

Wyniki uzyskane przy wykorzystaniu TFIDF:

```
------ Klaster: UnorderedNGram[kernel]
Kernel-HOWTO
Kernel-README
```

```
------ Klaster: UnorderedNGram[apache]
Apache-Overview-HOWTO
Unicode-HOWTO
Apache-Compile-HOWTO
```

Wykorzystane w teście fragmenty dzieł literackich są jedynie krótkimi ich wycinkami, zatem powtórzenia słów bądź całych fraz w różnych fragmentach jednego utworu zdarzają się rzadko i zakwalifikowanie takich fragmentów do wspólnego klastra stanowi bardziej wynik przypadku niż faktycznego podobieństwa. Wykryte podobieństwo wybranych dokumentów typu HOWTO ma natomiast pokrycie w ich tematyce i jest zgodne z oczekiwaniami.

6.3.2.2. Stemming

W celu określenia wpływu użytych algorytmów stemmingu na proces grupowania dokumentów, porównane zostały grupy klastrów utworzone dla zestawu filtrów `lowc-fixa-dl-stopl` oraz tego samego zestawu wzbogaconego dodatkowo o stemmer. Zarówno w przypadku języka angielskiego jak i języka polskiego zaobserwowano wzrost liczby poprawnie zgrupowanych dokumentów po dodaniu stemmera do listy filtrów. W przypadku języka polskiego poprawa była w większości testów bardzo znaczna i większa niż w przypadku języka angielskiego.

Przykład wzrostu liczby poprawnych klastrów oraz porównanie wyników działania stemmera Lovins i stemmera Egothor dla języka angielskiego przedstawiono poniżej na przykładzie wyników testu `test05`.

Brak filtrów (`_rtsai(length@0,n@1)_cos_hwh`):

```
------ Klaster: UnorderedNGram[NT]
Linux+NT-Loader
Linux+WinNT

------ Klaster: UnorderedNGram[LDAP]
LDAP-HOWTO
LDAP-Implementation-HOWTO

------ Klaster: UnorderedNGram[Terminal]
Xterminals
XDM-Xterm

------ Klaster: UnorderedNGram[cvsd]
Secure-CVS-Pserver
XFree86-R200
```

Po dodaniu podstawowych filtrów (`lowc-fixa-dl-stopl_rtsai(length@0,n@1)_cos_hwh`) uzyskano identyczną klasteryzację jak przy ich braku.

Ze stemmerem Lovins (`lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@1)_cos_hwh`):

```
------ Klaster: UnorderedNGram[bootsect]
Linux+NT-Loader
Linux+WinNT
LILO

------ Klaster: UnorderedNGram[modem]
Leased-Line
Linux-Modem-Sharing

------ Klaster: UnorderedNGram[ldap]
LDAP-HOWTO
LDAP-Implementation-HOWTO

------ Klaster: UnorderedNGram[158]
Xterminals
XDM-Xterm

------ Klaster: UnorderedNGram[uucp]
Sendmail+UUCP
Sendmail-Address-Rewrite
```

Ze stemmerem Egothor (`lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@1)_cos_hwh`):

```
------ Klaster: UnorderedNGram[bootsect]
Linux+NT-Loader
Linux+WinNT

------ Klaster: UnorderedNGram[geometry]
Large-Disk-HOWTO
LILO

------ Klaster: UnorderedNGram[modem]
Leased-Line
Linux-Modem-Sharing

------ Klaster: UnorderedNGram[ldap]
LDAP-HOWTO
LDAP-Implementation-HOWTO
```

```
------ Klaster: UnorderedNGram[terminal]
Xterminals
XDM-Xterm
```

```
------ Klaster: UnorderedNGram[uucp]
Sendmail+UUCP
Sendmail-Address-Rewrite
```

W przypadku innych zestawów dokumentów w języku angielskim zauważono podobny, umiarkowany wzrost liczby poprawnie znalezionych klastrów po dodaniu do listy filtrów jednego ze stemmerów. Stemmer Lovins i stemmer Egothor wydają się działać w przybliżeniu równie dobrze.

Testy potwierdziły duże znaczenie stemmingu przy przetwarzaniu dokumentów w języku polskim. Poprawa wyników grupowania szczególnie mocno zauważalna była przy stosowaniu n-gramów o n większym od 1. Jest to zgodne z przewidywaniami: dzięki stemmingowi, niektóre kombinacje słów, które bez niego potraktowane byłyby jak różne n-gramy, są uznawane za n-gramy identyczne. Liczba różnych n-gramów sprowadzanych w ten sposób do jednej postaci rośnie wraz z n, gdyż każde słowo wchodzące w skład n-gramu może podlegać odmianie, zwiększając łączną liczbę jego wariantów.

Poniżej przedstawiono przykłady wpływu użycia stemmera na grupowanie dokumentów w języku polskim (zestaw `test09`) przy wykorzystaniu TFIDF jako wagi.

Brak stemmera (`lowc-fixa-dl-stopl_rtsai(length@0,n@1)_cos_hwh`):

```
------ Klaster: UnorderedNGram[powiatu]
ust sam pow 1998 06 05 dzu 2001 142 1592 na 1 XII 2004.txt
ust sam gminny 1990 03 08  dzu 2001 142 1591 na 2004 12 01.txt

------ Klaster: UnorderedNGram[wyborczej]
ord PE na 2005 12 01.txt
ordynacja do sejmu senatu dzu 12 iv 2001 nr 46 poz 499.html
ordynacja PUE dzu 23 I 2004 nr 25 poz 219.htm
ord Prez na 2004 12 01.txt
```

Ze stemmerem Egothor (`lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@1)_cos_hwh`):

```
------ Klaster: UnorderedNGram[powiatać]
ust sam pow 1998 06 05 dzu 2001 142 1592 na 1 XII 2004.txt
ust sam gminny 1990 03 08  dzu 2001 142 1591 na 2004 12 01.txt
ust sam woj na 2005 01 11.txt

------ Klaster: UnorderedNGram[wyborczniy]
ord PE na 2005 12 01.txt
ordynacja do sejmu senatu dzu 12 iv 2001 nr 46 poz 499.html
ordynacja PUE dzu 23 I 2004 nr 25 poz 219.htm
ord Prez na 2004 12 01.txt
ust partie pol dzu2001 79 857 na 1 XII 2004.txt
```

Ze stemmerem SoundexPL (`lowc-fixa-dl-stopl-soundex(pl@true)_rtsai(length@0,n@1)_cos_hwh`):

```
------ Klaster: UnorderedNGram[1300]
ust sam pow 1998 06 05 dzu 2001 142 1592 na 1 XII 2004.txt
ust sam gminny 1990 03 08  dzu 2001 142 1591 na 2004 12 01.txt
ust sam woj na 2005 01 11.txt

------ Klaster: UnorderedNGram[6122]
ord PE na 2005 12 01.txt
ordynacja do sejmu senatu dzu 12 iv 2001 nr 46 poz 499.html
ordynacja PUE dzu 23 I 2004 nr 25 poz 219.htm
ord Prez na 2004 12 01.txt

------ Klaster: UnorderedNGram[4592]
przep wprow KK.txt
kk na 2004 12 01.txt
```

Przykłady pokazują, że użycie każdego ze stemmerów poprawiło jakość grupowania. W większości testów poprawa wyników po wprowadzeniu stemmingu była jeszcze silniejsza niż w przytoczonym powyżej przykładzie (np. w `test18`). Wystąpiły jednak również przypadki gdy wprowadzenie stemmingu nie zmieniło w żaden sposób znajdowanego zbioru klastrów (np. użycie stemmera Egothor dla `test16` (bigramy)). W powyższym przykładzie wykonania dla `test09` użycie SoundexPL zaskutkowało znalezieniem większej liczby poprawnych klastrów niż użycie stemmera Egothor. Podobnie było także m.in. dla bigramów w `test16`. W większości przypadków wyniki uzyskiwane za pomocą obu stemmerów były zbliżone i zdecydowanie lepsze niż gdy nie był wykorzystany żaden stemmer (patrz punkt 6.3.2.2.1).

Przy wykorzystaniu częstości jako wag i przestrzeni cech opartej o unigramy, wszystkie dokumenty zestawu `test09` zostały przypisane do jednego klastra – zarówno wtedy gdy nie były stosowane żadne filtry jak i wtedy, gdy stosowany był zestaw filtrów `lowc-fixa-dl-stopl`, a nawet po dodaniu stemmingu. Przy analogicznej konfiguracji w innych testach dla języka polskiego uzyskiwano zwykle grupowanie wszystkich lub prawie wszystkich dokumentów razem jedynie przy braku wykorzystania stemmera. Wyjątek stanowią zestawy zawierające wiele dokumentów stanowiących fragmenty dramatów lub zapisy rozmów (np. `test08`, w pewnym stopniu również `test16`) – w tych tekstach pojawiające się co kilka wierszy imiona mówiących osób prowadziły do wygenerowania podziałów opartych o te właśnie imiona. W innych testach dopiero użycie stemmera pozwoliło uzyskać w miarę dobre wyniki. Gdy wykorzystywane było ważenie cech algorytmem TFIDF, obserwowano znacznie mniejsze, aczkolwiek wciąż istotne różnice między wynikami generowanymi z użyciem

stemmera i bez jego pomocy. Stosowanie stemmingu dla dokumentów w języku polskim wydaje się zatem niezbędne bez względu na sposób ważenia cech.

6.3.2.2.1. Porównanie stemmerów Egothor i SoundexPL

Gdy wykorzystywane są unigramy, stemmer Egothor wydaje się mieć pewną przewagę nad SoundexPL, gdyż wykorzystanie SoundexPL prowadzi zwykle do znalezienia pewnej liczby poprawnych klastrów nie wykrytych gdy wykorzystywany jest stemmer Egothor, ale również do wzrostu liczby dokumentów przypisanych do klastrów w sposób niewłaściwy. W niektórych testach dawała się obserwować szczególnie silna tendencja SoundexPL do sprowadzania do identycznego kodu słów niepodobnych. Obserwowano wówczas powstawanie klastrów zawierających zupełnie różne dokumenty. Klaster taki jest obecny np. w wynikach `lowc-fixa-dl-stopl-soundex(pl@true)_rtsai(length@0,n@1)_cos_hwh` dla `test18`, ale znajdują się tam także klastry poprawne, znalezione z użyciem SoundexPL, a nie znajdowane z użyciem stemmera Egothor.

Zdecydowany spadek liczby dokumentów niepoprawnie przypisanych do klastrów zaobserwowano w przypadku SoundexPL przy przejściu od unigramów do bi- i trigramów. Już przy stosowaniu bigramów można na podstawie testów uznać algorytm SoundexPL za równie dobry jak Egothor (przynajmniej w zastosowaniu do grupowania dokumentów; przydatność w innych zastosowaniach, np. do wyszukiwania dokumentów pasujących do zadanych haseł prawdopodobnie jest mniejsza). Testy pozwalają zatem sądzić, że SoundexPL może nadawać się do zastosowań praktycznych.

Nasilenie trudności związanych z tworzeniem niepoprawnych klastrów przez algorytm SoundexPL okazało się w praktyce mniejsze od przewidywanego. Można przypuszczać, że algorytm ten wypada całkiem dobrze w porównaniu ze stemmerem Egothor dla języka polskiego ponieważ ten ostatni zawiera reguły wygenerowane na podstawie zbioru lematów, a nie rdzeni. W odróżnieniu zatem od np. stemmera Lovins często nie sprowadza on do identycznej postaci różnych słów o tym samym rdzeniu, a jedynie różne formy tego samego słowa. SoundexPL posiada wprawdzie ograniczoną zdolność obcinania długich przyrostków, ale prawdopodobnie nadrabia ten brak zdolnością do sprowadzania do jednego kodu różnych słów pochodzących od tego samego rdzenia gdy przyrostki są krótkie.

6.3.2.2.2. Stemmer Egothor w trybie iteracyjnym i nieiteracyjnym

W przypadku języka angielskiego, oba tryby pracy stemmera Egothor prowadziły w wykonanych testach do identycznych wyników grupowania (np. `test01`, `test05`). W przypadku języka polskiego, wyłączenie iteracyjnego trybu pracy stemmera powodowało niemal zawsze zauważalne pogorszenie wyników grupowania. Przykładowo, w `test18` otrzymano w przebiegu `lowc-fixa-dl-stopl-egothor(recursive@false)_rtsai(length@0,n@1)_cos_hwh` przedstawione poniżej klastry. Porównanie z przedstawionymi w punkcie 6.3.3.1 wynikami dla `test18/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@1)_cos_hwh` pozwala stwierdzić znacznie gorsze działanie stemmera w wersji nieiteracyjnej niż iteracyjnej:

```
------ Klaster: UnorderedNGram[lis]
lis_i_jaskolka.htm
szelmostwa_lisa_witalisa.htm
------ Klaster: UnorderedNGram[lew]
lew_i_zwierzeta.htm
lew_pokorny.htm
------ Klaster: UnorderedNGram[rak]
rak.htm
ryby_zaby_raki.htm
```

```
------ Klaster: UnorderedNGram[soczazk]
pan_soczewka_na_ksiezycu.htm
pan_soczewka_na_dnie_oceanu.htm
pan_soczewka_w_puszczy.htm
------ Klaster: UnorderedNGram[kacza]
kaczki.htm
kaczka dziwaczka.htm
```

Można przypuszczać, że słabość stemmera Egothor dla języka polskiego, zwłaszcza w wersji nieiteracyjnej, związana jest w znacznej mierze z wykorzystaniem jego wersji niekomercyjnej, wygenerowanej na podstawie stosunkowo małego zbioru uczącego.

6.3.3. Przestrzeń cech

Rodzaj przestrzeni cech, w której reprezentowane są dokumenty tekstowe bardzo mocno wpływa na wyniki ich grupowania. Zarówno wybór cech jak i ewentualna redukcja wymiaru ich przestrzeni pozwalają na kontrolowanie ziarnistości podziału zbioru dokumentów na grupy.

6.3.3.1. Długość i rodzaj n-gramów

Wpływ długości wykorzystanych n-gramów na wyniki grupowania zależy w dużej mierze od charakteru grupowanych dokumentów. W przypadku dokumentów krótkich (kilka kB), wykorzystanie unigramów dawało w testach prawie zawsze lepsze wyniki niż wykorzystanie dłuższych n-gramów. W przypadku krótkich dokumentów HTML zjawisko to było dodatkowo wzmocnione przez fakt nadania wysokich wag n-gramom zawartym w tytule strony: krótka treść dokumentu nie była w stanie zrównoważyć

wpływu tytułu. Ponieważ tytuły zwykle są krótkie, szansa powtórzenia się w różnych tytułach bi- lub trigramu jest mniejsza niż unigramu. Z drugiej strony, porównanie wyników otrzymanych dla krótkich dokumentów zawierających znaczniki HTML określające ich tytuły (`test18`) oraz tych samych dokumentów pozbawionych owych znaczników (`test19`) świadczy o tym, że korzystanie ze znaczników HTML może bardzo ułatwić znajdowanie poprawnych grup.

`test18/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@1)_cos_hwh:`

```
------ Klaster: UnorderedNGram[lis]
lis_i_jaskolka.htm
szelmostwa_lisa_witalisa.htm

------ Klaster: UnorderedNGram[lew]
lew_i_zwierzeta.htm
lew_pokorny.htm

------ Klaster: UnorderedNGram[rak]
rak.htm
ryby_zaby_raki.htm

------ Klaster: UnorderedNGram[soczać]
pan_soczewka_na_ksiezycu.htm
pan_soczewka_na_dnie_oceanu.htm
pan_soczewka_w_puszczy.htm

------ Klaster: UnorderedNGram[kacza]
kaczki.htm
kaczka dziwaczka.htm

------ Klaster: UnorderedNGram[chrzać]
chrzan.htm
cwikla.htm

------ Klaster: UnorderedNGram[ojcie]
monachomachia.htm
antymonachomachia.htm

------ Klaster: UnorderedNGram[sić]
szpak_i_sowa.htm
sowa.htm

------ Klaster: UnorderedNGram[ptas]
ptasi_mozg.htm
ptaszki_w_klatce.htm
```

`test18/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@2)_cos_hwh:`

```
------ Klaster: UnorderedNGram[opowiadać,drastyczny]
samotnosc.htm
opowiadania_drastyczne.htm

------ Klaster: UnorderedNGram[soczać,pan]
pan_soczewka_na_ksiezycu.htm
pan_soczewka_na_dnie_oceanu.htm
pan_soczewka_w_puszczy.htm

------ Klaster: UnorderedNGram[ignae,krasicka]
dewotka.htm
malarze.htm
filozof.htm
krasicki.htm
```

`test18/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@3)_cos_hwh:`

```
------ Klaster: UnorderedNGram[ojczyzna,miłośći,hy]
hymn_do_milosci_ojczyzny.htm
krasicki.htm
```

`test19/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@1)_cos_hwh:`

```
------ Klaster: UnorderedNGram[muł]
mul.txt
chory_mul.txt

------ Klaster: UnorderedNGram[honoratnia]
antymonachomachia.txt
monachomachia.txt

------ Klaster: UnorderedNGram[soczać]
pan_soczewka_w_puszczy.txt
pan_soczewka_na_ksiezycu.txt
pan_soczewka_na_dnie_oceanu.txt

------ Klaster: UnorderedNGram[rak]
rak.txt
ryby_zaby_raki.txt
```

```
   test19/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@2)_cos_hwh:
Nie znaleziono żadnych nietrywialnych klastrów.

   test19/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@3)_cos_hwh:
Nie znaleziono żadnych nietrywialnych klastrów.
```

Podobne wnioski dla krótkich dokumentów w języku angielskim można wyciągnąć na podstawie `test17`: wyniki uzyskane dla unigramów są niemal idealnie zgodne z oczekiwanym podziałem na grupy, zaś uzyskane z użyciem bigramów są nieco gorsze, aczkolwiek różnica nie jest tak znaczna jak w `test19`. Generalnie zauważono znacznie lepsze działanie bigramów dla dokumentów w języku angielskim niż dla dokumentów w języku polskim (co może wynikać ze swobodnego szyku języka polskiego).

Wyniki uzyskiwane dla bigramów oraz trigramów okazały się bardzo czułe na to, czy wykorzystywana była pełna przestrzeń cech, czy jej wymiar poddawany był redukcji. W pierwszym przypadku uzyskiwano zwykle niezbyt dobre wyniki, przedstawione poniżej. W drugim okazało się, że stosowanie poligramów korzystnie wpływa na proces grupowania (szczegóły przedstawiono w punkcie 6.3.3.3). Należy sądzić, że ta różnica wynika ze sposobu działania wykorzystanego klasteryzatora. Poniżej przedstawiono wyniki uzyskane dla przestrzeni cech nie poddanych redukcji wymiaru.

W przypadku dokumentów dłuższych niż kilka kilobajtów, wykorzystanie n-gramów dłuższych od jedności niekiedy prowadziło do poprawy wyników grupowania. Zjawisko to miało miejsce głównie dla zbiorów zawierających dokumenty dość podobne. W przypadku zbiorów dokumentów znacznie się różniących przejście od unido bigramów zwykle powodowało pogorszenie jakości grupowania nawet gdy wszystkie dokumenty były stosunkowo długie (np. `test06`, `test09`). Bardzo często już użycie bigramów powodowało przypisanie każdego dokumentu do oddzielnego klastra. Wykrywanie grup przez klasteryzator wyłącznie w oparciu o wejściowy zbiór punktów, bez korzystania ze z góry ustalonych wartości progowych mogłoby zapewne w wielu przypadkach poprawić wyniki uzyskiwane dla bigramów w stosunku do wyników dla unigramów. Niestety wykorzystane klasteryzatory nie posiadały takiej możliwości.

Sam wybór długości n-gramów można potraktować jako jeden ze sposobów określenia czułości klasteryzacji (krótsze n-gramy powodują powstawanie większych, bardziej ogólnych klastrów, dłuższe zaś powodują podział na węższe kategorie). Wy-

korzystanie bigramów miało najbardziej zauważalny wpływ na zmniejszenie czułości grupowania przede wszystkim wtedy, gdy nie był wykonywany stemming oraz gdy wykorzystywane były różne metody redukcji wymiaru przestrzeni cech. Przykładu dostarcza np. porównanie uzyskanych dla `test10` wyników `lowc-fixa-dl-stopl-egothor_rtsai(length@100,n@1)_cos_hwh` i `lowc-fixa-dl-stopl-egothor_rtsai(length@100,n@2)_cos_hwh`.

Stosowanie trigramów i dłuższych n-gramów oraz przestrzeni cech o niezredukowanym wymiarze niemal zawsze prowadziło do umieszczenia każdego dokumentu w osobnym klastrze albo do stworzenia bardzo niewielkiej liczby klastrów o więcej niż jednym elemencie. Po wykonaniu redukcji wymiaru przestrzeni cech w niektórych wypadkach uzyskiwano rozsądne (aczkolwiek bardzo drobnoziarniste) podziały przy użyciu trigramów (np. `test03`, `test18`). Dłuższe n-gramy okazały się raczej bezużyteczne.

Zarówno w przypadku języka polskiego jak i angielskiego różnice w wynikach grupowania wykonywanego za pomocą bigramów uporządkowanych i nieuporządkowanych były bardzo nieznaczne (przykłady m.in. w wynikach dla `test10`, `test16`).

6.3.3.2. Sposób ważenia cech

W niemal wszystkich testach ważenie cech za pomocą TFIDF okazało swoją wyższość nad ważeniem ich za pomocą częstości względnych. W wielu przypadkach, m.in. w przykładzie przytoczonym w punkcie 6.3.2.1, użycie TFIDF pozwoliło osiągnąć całkiem dobre wyniki grupowania nawet bez korzystania ze skomplikowanych filtrów tekstu, podczas gdy wyniki osiągane w identycznej konfiguracji z użyciem częstości jako wag były niepoprawne. Redukcja wymiaru przestrzeni cech pozwoliła w pewnych wypadkach uzyskiwać dość dobre wyniki również w sytuacji ważenia cech za pomocą częstości względnych (patrz punkt 6.3.3.3).

6.3.3.3. Wymiar

Okazało się, że redukcja wymiaru przestrzeni cech bardzo pozytywnie wpływa na wynik grupowania, powodując znajdowanie sensownych grup tam, gdzie wykorzystanie wszystkich wymiarów oryginalnej przestrzeni cech powodowało umieszczenie każdego dokumentu w osobnym klastrze. Trudno stwierdzić jednoznacznie, na ile poprawa ta wynika z zalet użytych algorytmów redukcji wymiaru przestrzeni cech, a na ile z wad wykorzystanych algorytmów klasteryzacji. Istnieją poważne podstawy by

sądzić, że wykorzystane klasteryzatory grupują dokumenty coraz słabiej wraz ze wzrostem wymiaru przestrzeni cech (patrz dyskusja w punkcie 6.3.1) i że właśnie dlatego redukcja wymiaru tej przestrzeni znacznie poprawia wyniki.

W większości testów, w których grupowanie umieszczało prawie każdy dokument w oddzielnym klastrze gdy wykorzystywana była pełna przestrzeń cech, po zredukowaniu liczby wymiarów do pomiędzy 100 a 1000 otrzymywano bardzo rozsądny podział na grupy. W większości testów dopiero redukcja wymiaru przestrzeni cech pozwalała uzyskiwać sensowne wyniki przy użyciu bigramów a czasem nawet trigramów.

Poniżej przedstawiono przykładowe wyniki uzyskane dla danych `test18` po redukcji wymiaru przestrzeni cech do 1000. Porównanie z przytoczonymi w punkcie 6.3.3.1 wynikami uzyskanymi bez redukcji wymiaru przestrzeni cech, wynoszącego 17175 w przypadku unigramów, 50136 w przypadku bigramów i 44462 w przypadku trigramów, pokazuje znaczny wzrost liczby znajdowanych poprawnych klastrów. Wzrost ten jest szczególnie silny w przypadku bigramów i trigramów. W wielu innych testach (np. `test19`) dopiero redukcja wymiaru przestrzeni cech pozwoliła w ogóle uzyskać jakiekolwiek nietrywialne klastry dla trigramów. Podobnie jak w przestrzeni o niezredukowanym wymiarze, klastry znalezione przy użyciu poligramów wyznaczają węższe grupy tematyczne niż klastry znalezione przy użyciu unigramów.

```
lowc-fixa-dl-stopl-egothor_rtsai(length@1000,n@1)_cos_hwh:

------ Klaster: UnorderedNGram[dziura]
dziura_w_moscie.htm
wrona_i_ser.htm

------ Klaster: UnorderedNGram[żółw]
zolw.htm
zolwie_i_krokodyle.htm

------ Klaster: UnorderedNGram[miłość]
quasi_una_fantasia.htm
opowiadania_drastyczne.htm

------ Klaster: UnorderedNGram[much]
za_krola_jelonka.htm
pajak_i_muchy.htm

------ Klaster: UnorderedNGram[lis]
lis_i_jaskolka.htm
szelmostwa_lisa_witalisa.htm

------ Klaster: UnorderedNGram[wyspa]
na_wyspach_bergamuta.htm
trzy_wesole_krasnoludki.htm

------ Klaster: UnorderedNGram[lew]
lew_i_zwierzeta.htm
lew_pokorny.htm
krasicki.htm

------ Klaster: UnorderedNGram[pan]
schizofrenia.htm
samotnosc.htm

------ Klaster: UnorderedNGram[rak]
rak.htm
ryby_zaby_raki.htm

------ Klaster: UnorderedNGram[wć]
wino_i_woda.htm
do_krola.htm

------ Klaster: UnorderedNGram[króleźć]
basn_o_korsarzu_palemonie.htm
kopciuszek.htm

------ Klaster: UnorderedNGram[balladać]
ballada.htm
druga_ballada_o_malej_ksiezniczce.htm

------ Klaster: UnorderedNGram[soczać]
pan_soczewka_na_ksiezycu.htm
```

```
pan_soczewka_na_dnie_oceanu.htm
pan_soczewka_w_puszczy.htm

------ Klaster: UnorderedNGram[kacza]
kaczki.htm
kaczka dziwaczka.htm

------ Klaster: UnorderedNGram[kot]
kot_w_butach.htm
panieneczka_z_pudeleczka.htm

------ Klaster: UnorderedNGram[chrzać]
chrzan.htm
cwikla.htm

------ Klaster: UnorderedNGram
[monachomachia]
monachomachia.htm
antymonachomachia.htm

------ Klaster: UnorderedNGram[sić]
szpak_i_sowa.htm
sowa.htm

------ Klaster: UnorderedNGram[chory]
chory_mul.htm
mul.htm

------ Klaster: UnorderedNGram[ptas]
ptasi_mozg.htm
ptaszki_w_klatce.htm
```

lowc-fixa-dl-stopl-egothor_rtsai(length@1000,n@2)_cos_hwh:

```
------ Klaster: UnorderedNGram[opowiadać,drastyczny]
quasi_una_fantasia.htm
zazdrosc.htm
schizofrenia.htm
samotnosc.htm
opowiadania_drastyczne.htm

------ Klaster: UnorderedNGram[ignae,krasicka]
pijanstwo.htm
do_krola.htm
dewotka.htm
malarze.htm
monachomachia.htm
antymonachomachia.htm
filozof.htm
krasicki.htm

------ Klaster: UnorderedNGram[soczać,pan]
pan_soczewka_na_ksiezycu.htm
pan_soczewka_na_dnie_oceanu.htm
pan_soczewka_w_puszczy.htm
```

lowc-fixa-dl-stopl-egothor_rtsai(length@1000,n@3)_cos_hwh:

```
------ Klaster: UnorderedNGram[fantasy,quas,una]
quasi_una_fantasia.htm
opowiadania_drastyczne.htm

------ Klaster: UnorderedNGram[ojczyzna,miłośći,hy]
hymn_do_milosci_ojczyzny.htm
krasicki.htm
```

Po wykonaniu testów w takiej samej konfiguracji ale z użyciem rozszerzonej wersji algorytmu SAI, uzyskano identyczne wyniki jak z użyciem wersji podstawowej tego algorytmu, z wyjątkiem unigramów, dla których otrzymano o jeden klaster mniej. W innych testach (np. `test01`) obie odmiany algorytmu również dawały praktycznie takie same wyniki.

W innych testach (m.in. `test05`, `test06`, `test13`, `test16`) uzyskano przy wykorzystaniu redukcji wymiaru przestrzeni cech podobne wyniki. Poprawa znajdowanej klasteryzacji była znaczna zarówno po wykorzystaniu metody SAI i ważeniu cech za

pomocą TFIDF jak i po wykorzystaniu redukcji wymiaru w oparciu o częstości względne i wykorzystaniu tych częstości jako wag. Dla większości zbiorów danych optymalne wyniki otrzymywano przy użyciu TFIDF i redukcji wymiaru przestrzeni cech za pomocą SAI do około 1000. W przypadku większych zbiorów danych (np. `test14`) lepsze wyniki osiągano gdy wymiar zredukowanej przestrzeni był nieco wyższy.

Po redukcji wymiaru przestrzeni cech uzyskiwano czasem dość dobre wyniki przy wykorzystaniu częstości jako wag (np. w `test01`). Trzeba zauważyć, że sam algorytm redukcji wymiaru przestrzeni cech odgrywał tu znaczną rolę, gdyż wybierane są w nim cechy charakteryzujące się w pewnych dokumentach bardzo wysoką częstością, a więc dające większą niż inne szansę na odróżnienie dokumentów między sobą.

6.3.4. Klasteryzacja

Testy pokazały, że oba zaimplementowane algorytmy klasteryzacji znajdują dla przebadanych zestawów dokumentów identyczne zbiory klastrów. Ze względu na sposób działania tych algorytmów, wykorzystanie metryk innych niż kosinusowa napotkało na wiele trudności.

6.3.4.1. Metryka

Ze względu na wymienione w punkcie 6.3.1 wady klasteryzatora, przystosowanego w zasadzie tylko do korzystania z metryki kosinusowej, testy w innych metrykach (zwłaszcza przyjmujących nieograniczony zbiór wartości) zwykle prowadziły do znalezienia nieprawidłowego podziału zbioru dokumentów na grupy. W przypadku metryki maksimum, zwykle bardzo znaczna część wszystkich dokumentów umieszczana była w pojedynczym klastrze (np. `test06`, `test18`).

Okazało się, że „metryka” Jacquarda działa nieco podobnie do metryki kosinusowej, ale z mniejszą siłą grupowania: zwykle dla unigramów znajdowany przez nią zbiór nietrywialnych klastrów stanowił podzbiór zbioru znajdowanego przy użyciu metryki kosinusowej (np. `test05`), zaś dla bigramów każdy dokument umieszczany był w oddzielnym klastrze. Po redukcji wymiaru przestrzeni cech do około 100-1000, wyniki grupowania stawały się całkiem dobre i zbliżone do wyników uzyskiwanych dla metryki kosinusowej (np. w `test06`).

Pozostałe metryki niemal zawsze powodowały umieszczenie każdego dokumentu w oddzielnym klastrze bez względu na długość n-gramów. Redukcja wymiaru przestrzeni cech nie zmieniała tego stanu rzeczy.

6.3.4.2. Klasteryzator

W punkcie 6.3.1 wymienione zostały niektóre ograniczenia wykorzystanych w projekcie klasteryzatorów. Dla porównania klasteryzatora HWH i zmodyfikowanego k-means wykonano testy jedynie w metryce kosinusowej. We wszystkich przeprowadzonych testach wyniki grupowania przez te klasteryzatory były identyczne, zarówno w przypadku przestrzeni cech o wymiarze zredukowanym, jak i niezredukowanym (np. `test06`, `test07`, `test11`).

Porównując wyniki otrzymywane dla dwóch zbiorów dokumentów zawierających w większości te same dokumenty, można dojść do wniosku, że dodanie nawet jednego dokumentu do zbioru może silnie wpłynąć na znajdowany zbiór klastrów. Odpowiedzialne są za to zarówno generowane wektory (dodatkowy dokument wpływa na wybór przestrzeni cech) jak i klasteryzator. Zestawy dokumentów `test11` i `test12` różnią się jedynie obecnością pliku `Dracula.txt` w drugim z nich. Dodanie tego pliku wpłynęło jednak na przydział do klastrów również innych plików (`Gullivers Travels.txt`).

```
   test11/lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@1)_cos_hwh:

------ Klaster: UnorderedNGram[gerd]
Andersen Fairy Tales.txt
Grimm Fairy Tales.txt

   test12/lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@1)_cos_hwh:

------ Klaster: UnorderedNGram[said]
Grimm Fairy Tales.txt
Andersen Fairy Tales.txt
Gullivers Travels.txt
Dracula.txt
```

6.3.5. Nadawanie klastrom nazw

W większości przypadków nazwy nadawane klastrom, zarówno składającym się z wielu dokumentów jak i składającym się z pojedynczego dokumentu, można uznać za dobre opisy wchodzących w ich skład tekstów. Wybrane przykłady poprawnie nazwanych klastrów znalezionych w różnych testach przedstawiono poniżej (zniekształcenie słów wchodzących w skład nazw stanowi skutek użycia stemmera). Wiele innych

można wskazać w wynikach przytoczonych w pozostałych punktach niniejszego rozdziału.

test05/lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@1)_cos_hwh:

```
------ Klaster: UnorderedNGram[ldap]
LDAP-HOWTO
LDAP-Implementation-HOWTO
```

test17/lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@2)_cos_hwh:

```
------ Klaster: UnorderedNGram[og,vorb]
libvorbis-1.0.1-i486-1.txt
vorbis-tools-1.0.1-i486-2.txt
```

test09/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@2)_cos_hwh:

```
------ Klaster: UnorderedNGram[wyborczniy,komisjać]
ord PE na 2005 12 01.txt
ordynacja do sejmu senatu dzu 12 iv 2001 nr 46 poz 499.html
ordynacja PUE dzu 23 I 2004 nr 25 poz 219.htm
ord Prez na 2004 12 01.txt
```

test18/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@1)_cos_hwh:

```
------ Klaster: UnorderedNGram[lis]
lis_i_jaskolka.htm
szelmostwa_lisa_witalisa.htm
```

Generalnie stosowanie dłuższych n-gramów prowadziło do wygenerowania lepszych nazw klastrów nawet wtedy, gdy równocześnie powodowało pogorszenie jakości grupowania. Za przykład może posłużyć opis grupy zawierającej dokument `rfc1952.txt` z zestawu `test06` generowany dla różnych długości n-gramów. Dla wszystkich n większych od 1 grupowanie umieszczało każdy dokument w osobnym klastrze, a więc zwiększenie długości n-gramów powodowało pogorszenie wyników grupowania. Równocześnie grupom (często jednoelementowym) nadawane były coraz lepsze nazwy. Wymieniony dokument stanowi definicję standardu kompresji gzip.

lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@1)_cos_hwh:

```
------ Klaster: UnorderedNGram[compres]
rfc1951.txt
rfc1952.txt
```

lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@2)_cos_hwh:

```
------ Klaster: UnorderedNGram[gzip,fil]
rfc1952.txt
```

lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@3)_cos_hwh:

```
------ Klaster: UnorderedNGram[fil,gzip,format]
rfc1952.txt
```

```
   lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@4)_cos_hwh:

------ Klaster: UnorderedNGram[specif,fil,gzip,format]
rfc1952.txt

   lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@5)_cos_hwh:

------ Klaster: UnorderedNGram[specif,fil,gzip,1996,format]
rfc1952.txt
```

Nazwy poprawnie utworzonych klastrów niekiedy okazywały się mylące, np. wtedy gdy jako nazwa klastra wybierany był n-gram nie występujący w niektórych należących do niego dokumentach. Jest to możliwe gdyż proces nadawania nazw jest w niniejszej implementacji niezależny od samego grupowania. Ewidentnie mylące nazwy klastrów pojawiają się w wynikach raczej rzadko. Przykłady:

```
   test05/lowc-fixa-dl-stopl-lovins_rtsai(length@0,n@1)_cos_hwh:

------ Klaster: UnorderedNGram[158]
Xterminals
XDM-Xterm

   test01/lowc-fixa-dl-stopl-egothor_rtsai(length@0,n@2)_cos_hwh:

------ Klaster: UnorderedNGram[xx(2)]
Kernel-README
```

Z rzadka dochodziło też do sytuacji, gdy różne klastry otrzymywały tę samą nazwę. Sytuacja taka może świadczyć o słabości klasteryzatora, który powinien był połączyć je w jeden. Przykładowo, wyniki `test02/lowc-fixa-dl-stopl_rtsai(length@0,n@2)_cos_hwh` zawierają następujące dwa klastry:

```
------ Klaster: UnorderedNGram[states,united]
constitution-1.txt

------ Klaster: UnorderedNGram[states,united]
constitution-2.txt
```

6.4. Wydajność

Wydajność algorytmów wydaje się być istotnym kryterium oceny ich przydatności w zastosowaniach praktycznych. Stworzona w ramach pracy aplikacja nie była optymalizowana pod kątem prędkości działania, ale zmierzone czasy wykonania poszczególnych testów pozwalają ocenić względną wydajność różnych algorytmów oraz skalowalność całego systemu.

6.4.1. Sposób testowania

6.4.1.1. Platforma i sposób uruchamiania programu

Testy wydajności wykonane zostały na maszynie z procesorem Athlon XP 1700+ wyposażonej w 256 MB pamięci operacyjnej. Program uruchamiano na maszynie wirtualnej Javy (JVM) zawartej w Sun JDK 1.4.2_04 działającej pod kontrolą systemu operacyjnego GNU/Linux z jądrem z rodziny 2.6.

Aby ograniczyć wpływ czynników zewnętrznych na czas wykonania obliczeń, na czas testów zamknięto zbędne aplikacje. Zdezaktywowano także partycję wymiany. Poprzez podanie parametrów `-Xms192m` i `-Xmx192m` na wierszu poleceń programu `java` wymuszono na JVM ustalenie rozmiaru pamięci dostępnej dla programu na 192 MB. Pozwoliło to na przyspieszenie jego działania oraz na zmniejszenie różnic czasu wykonania obliczeń między poszczególnymi ich uruchomieniami poprzez zmniejszenie znaczenia narzutów związanych z alokacją pamięci.

Maszyna wirtualna Javy firmy Sun korzysta z technologii HotSpot. Polega ona na analizie wykonywanego bytecode'u Javy i kompilacji pewnych jego fragmentów do kodu wykonywalnego konkretnej maszyny w celu przyspieszenia działania programu ([HotSpot]). Skutkiem takiego rozwiązania jest stopniowy wzrost prędkości działania programu w miarę wykonywania wielokrotnie tych samych fragmentów kodu. Pierwsze wykonanie pewnej sekcji kodu jest w całości interpretowane, a więc bardzo powolne, zaś kolejne iteracje skutkują wygenerowaniem skompilowanego kodu natywnego i jego coraz lepszą optymalizacją w następnych przebiegach.

Sun Java SDK zawiera dwie maszyny wirtualne Javy z technologią HotSpot. Noszą one nazwy Client VM oraz Server VM i korzystają z nieco innych algorytmów optymalizacji kodu. Pierwsza z nich jest używana domyślnie i została dostosowana do programów uruchamianych z rzadka, nie wymagających dużej mocy obliczeniowej. Optymalizuje ona kod pod kątem szybkiego startu aplikacji i małych narzutów w początkowej fazie jej działania. Server VM przeznaczona jest dla zadań intensywnych obliczeniowo i działających przez długi czas. Uruchamianie aplikacji i jej działanie zaraz po starcie są znacznie wolniejsze niż w przypadku Client VM, ale za to stosowane są bardziej złożone algorytmy optymalizacji kodu, które powodują że po pewnym czasie prędkość działania programu zdecydowanie przekracza jego prędkość osiąganą przy użyciu pierwszej z wymienionych JVM.

Omówione wyżej cechy maszyny wirtualnej Sun JVM zostały uwzględnione w procedurze testowej. Przede wszystkim, aby pomiary czasu można było uznać za wiarygodne i powtarzalne, po uruchomieniu programu procedura obliczeniowa uruchamiana była kilkukrotnie w celu „rozgrzania" JVM i pozwolenia technologii HotSpot na zoptymalizowanie odpowiednich fragmentów kodu. Doświadczalnie stwierdzono, że czasy kolejnych uruchomień algorytmu w przybliżeniu wyrównują się po około trzech do pięciu uruchomieniach w przypadku Client VM i po około dziesięciu uruchomieniach w przypadku Server VM i są znacznie krótsze od czasu pierwszego przebiegu algorytmu po uruchomieniu programu. Ustabilizowany czas wykonania był w pierwszym przypadku krótszy o około 20%, zaś w drugim nawet do prawie czterech razy krótszy od czasu zmierzonego przy pierwszym wykonaniu obliczeń.

Po przeprowadzeniu pewnej liczby wstępnych prób, do testów wybrano maszynę wirtualną Client VM (ze względu na mniejszą liczbę iteracji programu potrzebnych do jej „rozgrzania"). W przypadku obu maszyn istnieje możliwość wyłączenia w ogóle technologii HotSpot i korzystania wyłącznie z interpretera bytecode'u, bez kompilacji do kodu wykonywalnego. Pomiary wykazały, że uruchamianie programu właśnie w ten sposób daje najmniejsze różnice czasów poszczególnych przebiegów, ale czasy te są około 10 razy dłuższe niż przy włączonym HotSpot. Z tego względu zrezygnowano z użycia tego trybu wykonania do celów testowych.

Dla „rozgrzania" JVM, po uruchomieniu programu wykonywano obliczenia w zaplanowanej konfiguracji dziesięć razy po kolei. Uśrednione czasy przebiegów 6-10 posłużyły do stworzenia wykresów przedstawionych w dalszej części rozdziału. Do wydobycia mierzonych czasów z plików z wynikami i obliczenia wartości średnich wykorzystano polecenia powłoki oraz `awk`.

Wielokrotne wykonywanie tych samych obliczeń oprócz optymalizacji kodu przez technologię HotSpot powoduje również zapamiętanie w buforach systemu operacyjnego zawartości plików dyskowych wykorzystywanych przez program, a więc zmniejsza opóźnienia związane z dostępem do dysku, zakłócające pomiar czasu właściwych obliczeń.

Po uwzględnieniu wymienionych wyżej czynników, do testów wydajności uruchamiano program następującym wierszem poleceń:

```
java -Xms192m -Xmx192m edu/mgr/GUI --run 10 > ../tests-results/x/y
```

6.4.1.2. Pomiar czasu

Czas wykonania mierzony jest wewnątrz aplikacji przy pomocy klasy `StopWatch`, wykorzystującej standardowe metody pomiaru czasu w Javie: klasę `Date` i metodę `currentTimeMillis()` klasy `System`. W ten sposób mierzony może być tylko czas rzeczywisty, a nie czas procesora zużyty przez pewien fragment kodu. Pomiar czasów procesora spędzonych w trybie użytkownika i systemu jest możliwy za pomocą zewnętrznych narzędzi (np. polecenia `time`), ale ich użycie w przypadku aplikacji Javy może prowadzić do mylnych wyników, gdyż czas startu JVM jest znaczny i wpływa w istotny sposób na mierzony całkowity czas działania programu. Profiler hprof współpracujący z maszyną wirtualną Javy umożliwia pomiar stosunku czasów spędzonych przez program w różnych partiach kodu, ale nie udostępnia bezwzględnych długości zmierzonych odcinków czasu.

Znaczenie poszczególnych mierzonych odcinków czasu, których długości są wypisywane wraz z wynikami działania programu, jest następujące:

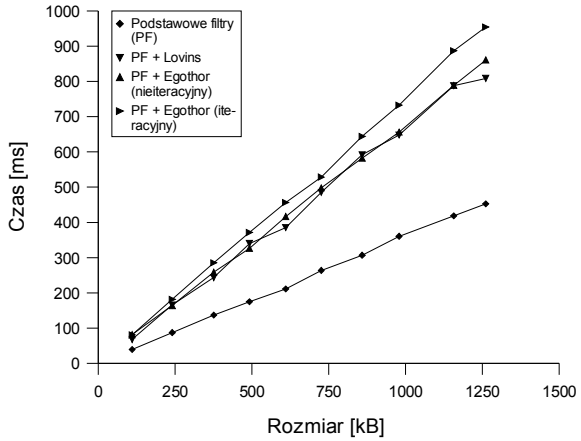
- Filtry: łączny czas zużyty przez wszystkie filtry.
- Konwerter: łączny czas działania filtrów, znalezienia przestrzeni cech i redukcji jej wymiaru oraz generacji współrzędnych. Zawiera w sobie czas ładowania dokumentów z dysku i tworzenia struktur danych reprezentujących je w pamięci oraz czas wymieniony w rubryce *Filtry*.
- Klasteryzator: czas działania klasteryzatora.

Mierzone czasy obejmują jedynie obliczenia i nie uwzględniają czynności pomocniczych jak wypisywanie wyników itp.

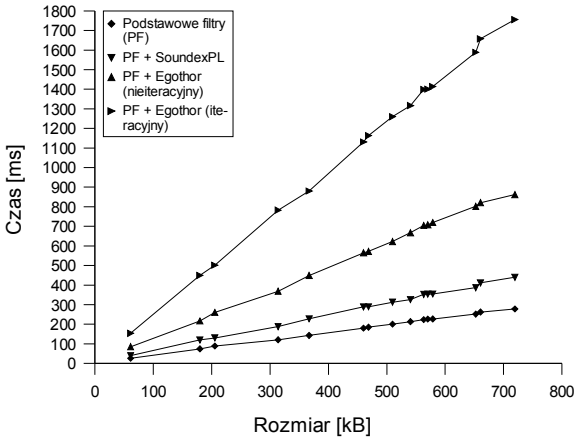
6.4.2. Stemming

Celem niniejszego testu było zbadanie zależności czasu wykonania poszczególnych algorytmów stemmingu od rozmiaru danych wejściowych. Pomiary wykonano dla zestawów dokumentów w językach angielskim (`test13`) i polskim (`test19`). Procedura testowa wyglądała w obu przypadkach identycznie: utworzono kopie odpowiednich katalogów z danymi i po wykonaniu testów dla wszystkich plików wykonywano serię kroków polegających na usunięciu pewnej ustalonej liczby plików i powtórzeniu pomiarów. Pliki usuwano w kolejności alfabetycznej nazw. Wyniki uzyskane w każdym kroku zawarte są w plikach o nazwach `perf_*_NNN`, gdzie `NNN`

oznacza liczbę plików uzyskaną w danym kroku. Wyniki działania skryptu `file-stats` na zestaw danych uzyskany w każdym kroku zapisano odpowiednio w plikach o nazwie `NNN`.



Wykres 1: Porównanie wydajności stemmerów dla języka angielskiego



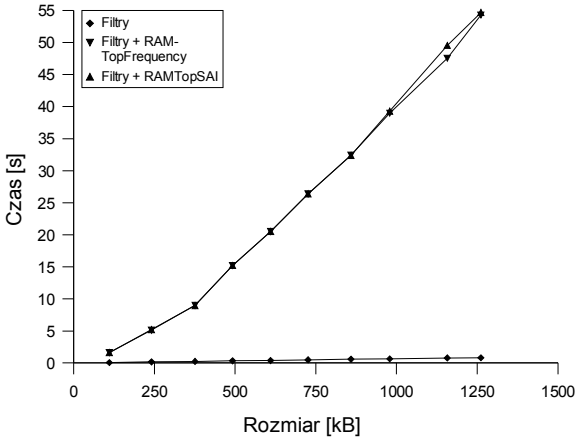
Wykres 2: Porównanie wydajności stemmerów dla języka polskiego

Na wykresach pokazano czasy wykonania typowych stemmerów dla języków angielskiego i polskiego oraz łączny czas działania podstawowych filtrów wykonywanych przed stemmingiem: zamiany liter na małe, usunięcia apostrofów pełniących funkcję cudzysłowów, wykrycia języka i zastosowania stop-listy. Czasy przedstawione są w funkcji rozmiaru surowych danych wejściowych. Ze względu na różną ilość słów usuwanych przez algorytm stop-listy, rozmiar danych ostatecznie pojawiających się na wejściu stemmerów mógł się różnić w zależności od języka. Ponieważ jednak stemming wykonuje się prawie zawsze już na dokumentach przetworzonych przez stop-listę, pokazanie czasu obliczeń właśnie w funkcji rozmiaru początkowych danych, a nie danych już częściowo przetworzonych uznano za najbardziej właściwe z punktu widzenia zastosowań praktycznych.

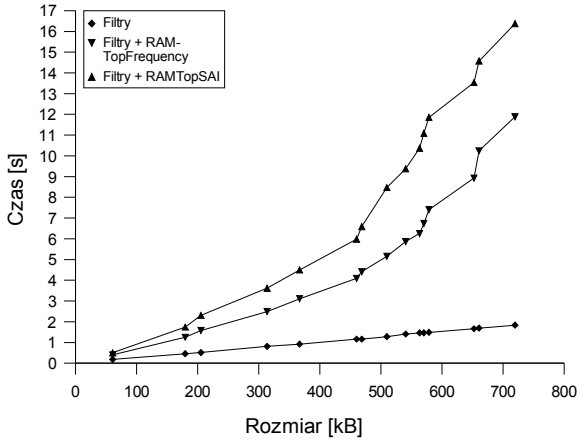
Wyniki pozwalają stwierdzić, że dla języka angielskiego stemmer Egothor w trybie nieiteracyjnym jest równie szybki jak stemmer Lovins, zaś jego wersja iteracyjna jest niewiele wolniejsza. Inaczej jest w przypadku języka polskiego. Wersja iteracyjna wykonuje się około 2,5 raza dłużej niż nieiteracyjna, ponadto nawet wersja nieiteracyjna jest zdecydowanie wolniejsza niż dla języka angielskiego. Stanowi to zapewne wynik bardziej złożonych reguł stemmingu koniecznych do odtworzenia zasad polskiej gramatyki oraz faktu, że w języku polskim znacznie więcej słów niż w języku angielskim wymaga więcej niż jednej iteracji. Algorytm SoundexPL okazał się zgodnie z przewidywaniami zdecydowanie szybszy od obu trybów działania stemmera Egothor.

6.4.3. Generacja współrzędnych

Czasy wykonania algorytmów generacji współrzędnych zaimplementowanych w klasach `RAMTopSAI` i `RAMTopFrequency` przy braku redukcji wymiaru przestrzeni cech przedstawiono na wykresach 3 i 4. Dla porównania zamieszczono na nich również czas działania filtrów (typowy zestaw: zamiana liter na małe, usunięcie apostrofów pełniących funkcję cudzysłowów, wykrycie języka (nie było w zasadzie konieczne), zastosowanie stop-listy i stemming algorytmem Lovins w przypadku `test13` i algorytmem Egothor w przypadku `test19`). Sposób generacji zbiorów danych do testów z wyjściowych zbiorów danych był identyczny jak opisany w punkcie 6.4.2.



Wykres 3: Wydajność generacji współrzędnych bez redukcji wymiaru przestrzeni cech (`test13`)

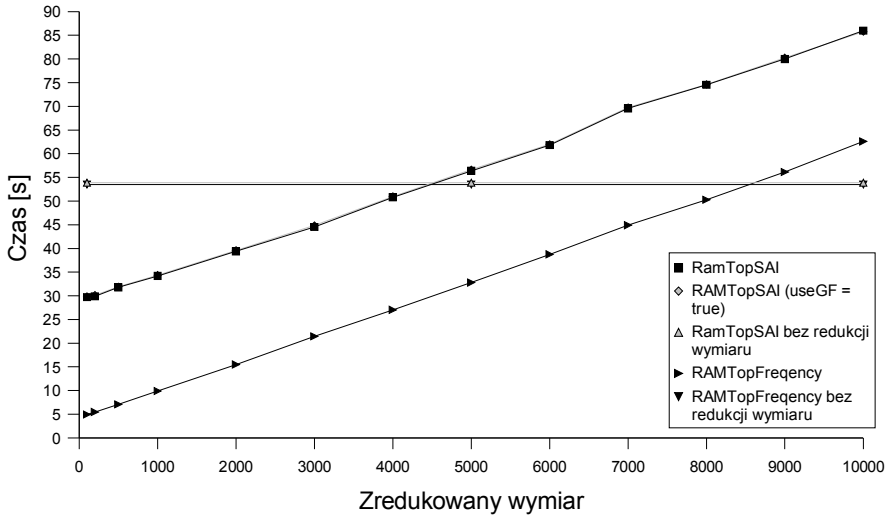


Wykres 4: Wydajność generacji współrzędnych bez redukcji wymiaru przestrzeni cech (`test19`)

Przykłady ilustrują dwie główne klasy zestawów danych testowych. Odróżnia je stosunek czasów działania algorytmów implementowanych przez `RAMTopSAI` i `RAMTopFrequency`. Różnica między tymi czasami jest duża dla testów, w których wysoki jest stosunek wymiaru przestrzeni cech d do rozmiaru danych s, np. dla standardowych filtrów i przestrzeni unigramów: `test19` ($d/s \approx 21{,}8$ [1/kB]), `test17` ($d/s \approx 22{,}1$ [1/kB])), zaś mała dla zbiorów danych, dla których stosunek ten jest niski (np. `test13` ($d/s \approx 8{,}64$ [1/kB]), `test09` ($d/s \approx 3{,}8$ [1/kB]). Związek różnicy tych czasów ze stosunkiem d/s prawdopodobnie wynika z konieczności zliczania dokumentów zawierających każde słowo w celu obliczenia współczynnika TFIDF. Oczywiście w każdym przypadku czas obliczeń dla ważenia cech za pomocą TFIDF jest dłuższy niż dla ich ważenia za pomocą częstości, ale dla zestawów takich jak `test13` różnica jest prawie niezauważalna. W sytuacji wysokiego stosunku d/s obserwować można dużo szybszy wzrost czasu obliczeń wraz ze wzrostem rozmiaru danych niż gdy stosunek ten jest niski.

6.4.4. Redukcja wymiaru przestrzeni cech

W teście porównano czas generacji współrzędnych w przypadku stosowania pełnej przestrzeni cech oraz w przypadku redukcji jej wymiaru. Testy przeprowadzono dla przestrzeni cech opartych o unigramy, jako zbiór danych testowych wykorzystując `test13`. Wykorzystano identyczny zestaw filtrów jak w punkcie 6.4.3. Niezredukowany wymiar przestrzeni cech wyniósł 10899.



Wykres 5: Wydajność generacji współrzędnych z redukcją wymiaru przestrzeni cech

Czasy generacji współrzędnych bez redukcji wymiaru cech nie we wszystkich testach były tak zbliżone dla ważenia częstościami i TFIDF jak na przedstawionym wykresie. Zagadnienie to omówione zostało w punkcie 6.4.3.

Porównanie czasów generacji współrzędnych z redukcją wymiaru przestrzeni cech i bez niej (poziome linie) pokazuje, że nawet dla stosunkowo słabej redukcji wymiaru, jej przeprowadzenie skraca łączny czas obliczeń. W przypadku metody SAI (klasa `RAMTopSAI`) do skrócenia czasu w stosunku do obliczeń bez redukcji wymiaru przestrzeni cech konieczna jest silniejsza redukcja wymiaru niż w przypadku redukcji wykonywanej przez klasę `RAMTopFrequency`. Narzut związany z wykorzystaniem rozszerzonej wersji algorytmu SAI (`useGF = true`) jest tak znikomy, że na wykresie 5 linie odpowiadające obu wersjom pokrywają się.

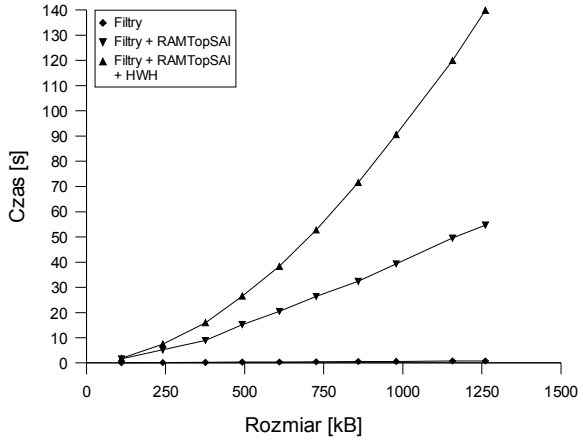
Prawdopodobnie nieznaczne przyspieszenie podstawowej wersji SAI (a przez to zwiększenie różnicy między nią a wersją rozszerzoną) można by uzyskać poprzez użycie w implementacji jedynie zbioru przechowującego wszystkie cechy występujące w zbiorze dokumentów zamiast odwzorowania przechowującego również liczbę wystąpień każdego z nich jak ma to miejsce w stworzonej implementacji. W ten sposób

dla każdego słowa dokumentu można by uniknąć jednej operacji zwiększenia licznika i być może kilku operacji dereferencji wskaźnika.

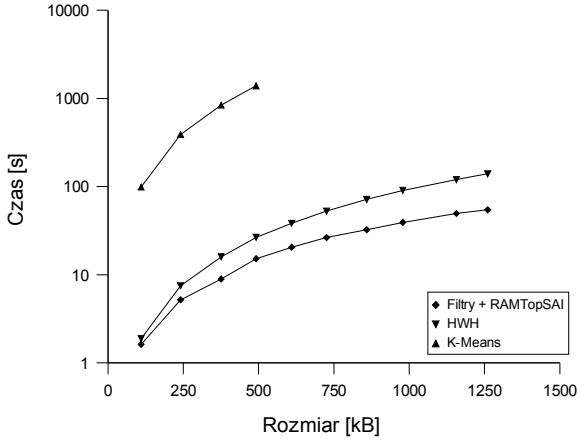
Subiektywna ocena wyników grupowania przedstawiona w punkcie 6.3.3.3 wykazała, że jakość wyników uzyskanych dla przestrzeni o nawet dość silnie zredukowanym wymiarze jest całkiem dobra, a ze względu na wykorzystany algorytm klasteryzacji, często wręcz lepsza niż dla przestrzeni cech o wymiarze nie zredukowanym. Redukcja wymiaru przestrzeni cech pozwala zatem uzyskiwać dostatecznie dobre wyniki przy jednoczesnym skróceniu czasu obliczeń.

6.4.5. Klasteryzacja

Test wydajności klasteryzatorów wykonano w celu porównania dwóch algorytmów zaimplementowanych w klasie `HwhKMeans` oraz zestawienia czasów ich działania z czasami działania filtrów oraz generatorów współrzędnych. Wykorzystano zestaw danych `test13`, postępując z nim w identyczny sposób jak w punkcie 6.4.2. Ze względu na niezwykle długi czas obliczeń klasteryzatora w wersji k-means, pomiary czasu jego działania wykonano tylko dla kilku najmniejszych zbiorów danych.



Wykres 6: Porównanie wydajności klasteryzatora HWH i generacji współrzędnych oraz filtrowania



Wykres 7: Porównanie wydajności klasteryzatorów HWH i k-means

Jak widać, czas zużywany przez algorytm HWH rośnie wraz z rozmiarem danych znacznie szybciej niż czas zużywany na filtrowanie tekstu i generację współrzędnych. W przypadku algorytmu k-means czasy wykonania są o dwa rzędy wielkości dłuższe niż czasy zużywane przez metodę HWH, co czyni tę implementację k-means praktycznie bezużyteczną. Wykorzystane testowe wersje klasteryzatorów okazały się zatem najwolniejszym elementem systemu.

6.5. Podsumowanie testów

Testy, w których oceniana była jakość grupowania dokumentów, pozwoliły na zbadanie wpływu poszczególnych algorytmów na końcowy wynik działania programu. Potwierdzona w nich została konieczność stosowania wstępnego przetwarzania tekstu, w szczególności stop-listy oraz stemmingu. Zaobserwowano znacznie większy wpływ stemmingu na wyniki grupowania dokumentów w języku polskim niż dokumentów w języku angielskim. Algorytm SoundexPL okazał się skuteczny i porównywalny z wersją niekomercyjną stemmera Egothor w przypadku stosowania przestrzeni cech opartej o bigramy i trigramy, lecz ustępuje temu stemmerowi w przypadku stosowania unigramów.

Eksperymentalnie stwierdzono, że stosowanie wartości TFIDF jako wag niemal zawsze skutkuje lepszymi wynikami niż stosowanie w roli wag częstości n-gramów. Ze względu na problemy związane z wykorzystaniem eksperymentalnej i nieprzetestowanej wersji klasteryzatora, nie było możliwe przeprowadzenie w pełni miarodajnych testów badających wpływ redukcji wymiaru przestrzeni cech na jakość generowanych wektorów. W przeprowadzonych testach redukcja wymiaru przestrzeni cech do około 100-1000 niemal zawsze powodowała poprawę wyników. Jest mało prawdopodobne, aby podobne wyniki zostały uzyskane również przy wykorzystaniu innych klasteryzatorów.

Przeprowadzone testy pokazały, że w przypadku stemmera Egothor, stemming dla języka polskiego wymaga znacznie większego nakładu obliczeń niż dla języka angielskiego. W przypadku języka polskiego obserwowane były również znacznie większe niż w przypadku języka angielskiego różnice w wydajności między wersją iteracyjną i nieiteracyjną tego stemmera. Algorytm SoundexPL okazał się znacznie szybszy od obu wymienionych trybów działania stemmera Egothor.

Pomiary wydajności redukcji wymiaru przestrzeni cech prowadzą do wniosku, że dla metody SAI i ważenia cech za pomocą TFIDF już redukcja wymiaru tej przestrzeni o około 60% powoduje skrócenie łącznego czasu obliczeń. Przy ważeniu cech za pomocą częstości względnych i redukcji wymiaru przez porządkowanie według maksimum sumy częstości względnych, wystarczająca jest redukcja wymiaru o zaledwie około 20%. Dalsza redukcja umożliwia znaczne (w przypadku ważenia cech za pomocą częstości – nawet kilkukrotne) skrócenie czasu obliczań. W testach oceny jakości

grupowania wyniki uzyskiwane nawet przy dziesięciokrotnym zmniejszeniu wymiaru przestrzeni cech były dość dobre. Redukcję wymiaru przestrzeni cech można zatem uznać za skuteczną metodę zwiększającą efektywność obliczeń przy niewielkim tylko pogorszeniu końcowych wyników. Często obserwowano wręcz poprawę wyników po redukcji wymiaru, być może w pewnym stopniu wynikającą z ograniczeń klasteryzatorów i ich słabego działania w przestrzeniach o wysokiej liczbie wymiarów.

Zmierzona wydajność klasteryzatorów potwierdziła ich testowy status. Zwłaszcza implementacja metody k-means charakteryzuje się bardzo powolnym działaniem, wykluczającym ją w zasadzie z zastosowań praktycznych. Wbrew oczekiwaniom okazała się ona znacznie wolniejsza od metody HWH. Oba klasteryzatory wykazywały szybki i silnie nieliniowy wzrost czasu obliczeń wraz ze wzrostem rozmiaru danych wejściowych.

Podsumowując, można stwierdzić że zaimplementowane algorytmy pozwalają na uzyskiwanie zadowalających wyników grupowania. Poprzez dobór odpowiednich parametrów algorytmów możliwe jest kontrolowanie ziarnistości znajdowanych podziałów zbioru dokumentów. Interesującym rozwinięciem przeprowadzonych testów mogłoby być ponowne ich przeprowadzenie z wykorzystaniem innych klasteryzatorów i porównanie wyników. Testy wydajności pokazały, że wstępne przetwarzanie dokumentów stanowi najkrócej trwającą część całego procesu grupowania, zaś najwięcej czasu zużywane jest na generację współrzędnych i klasteryzację. Te dwa etapy powinny zatem zostać rozpatrzone w pierwszej kolejności w celu znalezienia możliwych optymalizacji.

7. Podsumowanie

Stworzona w ramach niniejszej pracy aplikacja pozwoliła potwierdzić w praktyce skuteczność rozpatrywanych metod tworzenia reprezentacji dokumentów tekstowych w przestrzeni wektorowej i ich grupowania. Zbadane zostały między innymi: wpływ stosowanych metod wstępnego przetwarzania dokumentów, a zwłaszcza stemmingu, na przebieg procesu klasteryzacji oraz znaczenie właściwego wyboru przestrzeni cech dla poprawnej reprezentacji dokumentów. W testach potwierdzono także skuteczność metody SAI redukcji wymiaru przestrzeni cech. Modularna budowa opartego o system wtyczek programu pozwala na jego wykorzystanie w celu porównywania ze sobą zarówno algorytmów już zaimplementowanych jak i na dodanie obsługi nowych, być może jeszcze nie znanych w chwili jego pisania algorytmów.

Testy potwierdziły możliwość wykorzystania w praktyce algorytmu SoundexPL jako metody stemmingu dla języka polskiego. Przedstawiona w pracy wersja tego algorytmu posiada kilka słabych punktów, które obecnie utrudniają skuteczne jego wykorzystanie w połączeniu z przestrzenią cech o bazie unigramów. Można oczekiwać, że dalsze badania pozwolą na przezwyciężenie tych trudności i stworzenie wersji SoundexPL mogącej konkurować z innymi dotychczas używanymi algorytmami.

Równocześnie z niniejszą pracą rozwijane były prace magisterskie Michała Pawluczuka i Grzegorza Kolendo. Połączenie ostatecznych wersji modułów rozwijanych w ramach każdej z nich powinno pozwolić na stworzenie systemu grupującego dokumenty tekstowe pobierane bezpośrednio z internetu, np. na podstawie wpisanego hasła przekazywanego do wyszukiwarek internetowych. Wykorzystanie finalnej wersji klasteryzatorów zamiast dostępnej w momencie pisania niniejszej pracy wersji testowej powinno zdecydowanie poprawić uzyskiwane wyniki grupowania i doprowadzić system do pełnej użyteczności.

Dodatkowe uwagi

Słownik wyrazów bliskoznacznych języka angielskiego (`data/Thesaurus/en`) podlega następującej licencji:

```
This software and database is being provided to you, the LICENSEE, by
Princeton University under the following license.  By obtaining, using
and/or copying this software and database, you agree that you have
read, understood, and will comply with these terms and conditions.:

Permission to use, copy, modify and distribute this software and
database and its documentation for any purpose and without fee or
royalty is hereby granted, provided that you agree to comply with
the following copyright notice and statements, including the disclaimer,
and that the same appear on ALL copies of the software, database and
documentation, including modifications that you make for internal
use or for distribution.

WordNet 1.7 Copyright 2001 by Princeton University.  All rights reserved.
THIS SOFTWARE AND DATABASE IS PROVIDED "AS IS" AND PRINCETON
UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, PRINCETON
UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANT-
ABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE
OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT
INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR
OTHER RIGHTS.

The name of Princeton University or Princeton may not be used in
advertising or publicity pertaining to distribution of the software
and/or database.  Title to copyright in this software, database and
any associated documentation shall at all times remain with
Princeton University and LICENSEE agrees to preserve same.
```

Słownik wyrazów bliskoznacznych języka polskiego (`data/Thesaurus/pl`) podlega licencji GNU GPL.

Niniejsze oprogramowanie wykorzystuje kod stworzony w ramach Projektu Egothor (http://www.egothor.org/).

Autorem korpusu 20 Newsgroups jest Tom Mitchell, ze School of Computer Science, Carnegie Mellon University.

Zawartość płyty

Dołączona płyta CD zawiera:

- Kod źródłowy i skompilowaną wersję aplikacji (katalog `src`)
- Treść pracy magisterskiej w formatach OpenOffice.org oraz PDF (`dokumentacja`)
- Dane wykorzystane w testach (`tests`); Zestawy danych zawierające więcej niż 500 plików zostały umieszczone w archiwach `.tar.gz`, gdyż przeglądanie znajdujących się na płycie CD katalogów zawierających tak wiele plików mogłoby być zbyt powolne.
- Wyniki testów (`tests-results`)

Bibliografia

[SW]: W3C, *W3C Semantic Web*, http://www.w3.org/2001/sw/

[Jain]: Anil K. Jain, Narasimha Murty, Patrick J. Flynn, *Data Clustering: A Review*, ACM Computing Surveys 3, 1999

[Dominich]: Sándor Dominich, *Paradox-free Formal Foundation of Vector Space Model*, 25th Annual International Conference on Research and Development in Information Retrieval of the ACM SIGIR International Workshop Mathematical Formal Methods in Information Retrieval MF/IR '02, 2002

[Deerwester]: Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman, *Indexing By Latent Semantic Analysis*, Journal of the American Society For Information Science 41, 1990

[Porter, The Lovins stemmer]: Martin F. Porter, *The Lovins stemming algorithm*, http://snowball.tartarus.org/lovins/stemmer.html

[Porter]: Martin F. Porter, *An algorithm for suffix stripping*, Program 14(3), 1980

[Snowball]: Martin F. Porter, *Snowball Homepage*, http://snowball.tartarus.org/

[Bialecki]: Andrzej Bialecki, *Stempel - Algorithmic Stemmer for Polish Language*, http://getopt.org/stempel/index.html

[Weiss]: Dawid Weiss, *Stempelator: A Hybrid Stemmer for the Polish Language*, Poznań University of Technology Research Report RA-002/05, 2005

[Bartnicka]: Barbara Bartnicka, *Język i my*, tom 3, rozdz. 3 i 4, ISBN 83-02-04071-1

[Russell]: Robert C. Russell, *U.S. Patent 1,261,167*, 1918, http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=/netahtml/search-bool.html&r=1&f=G&l=50&co1=AND&d=o179&s1=1,261,167.WKU.&OS=PN/1,261,167&RS=PN/1,261,167

[Prosise]: Jeff Prosise, *No Matter How You Spell It, Soundex Finds It*, PC Magazine May 30, 1995

[Mokotoff]: Gary Mokotoff, *Soundexing and Genealogy*, http://www.avotaynu.com/soundex.html

[WordNet]: Cognitive Science Laboratory, Princeton University, *WordNet Homepage*, http://wordnet.princeton.edu/

[Salton]: Gerard Salton, Michael J. McGill, *Introduction to Modern Information Retrieval*, tom 1, rozdz. 2.A, 3.B, 5.A, 6, ISBN 0-07-054484-0

[Mirkin]: Boris Mirkin, *Mathematical Classification and Clustering*, tom 1, rozdz. 2.3.1 i 3.2.2, ISBN 0792341597

[PCA]: Wikipedia, *Principal Component Analysis*, http://en.wikipedia.org/wiki/Principal_components_analysis

[Koren]: Yehuda Koren, Liran Carmel, *Visualization of Labeled Data Using Linear Transformations*, Proceedings of IEEE Information Visualization 2003 (InfoVis'03), 2003

[LINGO]: Stanisław Osiński, *An algorithm for clustering of web search results*, 2003

[Späth]: Helmuth Späth, *Cluster Analysis Algorithms*, tom 1, rozdz. 3.2, ISBN 0-85312-141-9

[Ester]: Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996

[Knuth]: Donald E. Knuth, *Sztuka programowania*, tom 3, rozdz. 5.4, ISBN 83-204-2554-9

[HotSpot]: Sun Microsystems, *Java HotSpot Technology Documentation*, http://java.sun.com/products/hotspot/