POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI
INSTYTUT INFORMATYKI

PRACA DYPLOMOWA MAGISTERSKA

Rafał Jońca

SYSTEM WERYFIKACJI KOMENTARZY NA STRONACH WWW USPRAWNIAJĄCY PRACĘ MODERATORA

Promotor:
dr inż. Piotr FABIAN

Praca wykonana na kierunku
INFORMATYKA o specjalności
bazy danych, sieci i systemy komputerowe

GLIWICE 2006

Za pomoc w zebraniu publikacji dotyczących klasyfikacji tekstów dziękuję mgr Iwonie Żak. Firmie NETPRO dziękuję za udostępnienie kilkunastu tysięcy komentarzy z dawnego serwisu gry.pl oraz za hosting demonstracyjnej wersji systemu. Promotorowi dziękuję za pomoc okazaną przy korekcie i odzyskaniu materiałów po utracie dysku twardego.

SPIS TREŚCI

SPIS TABEL

SPIS RYSUNKÓW

1. WSTĘP

1.1. CEL PRACY

1.1.1. Dlaczego weryfikacja komentarzy jest potrzebna?

Stale zwiększa się liczba witryn internetowych oferujących komentowanie różnego rodzaju artykułów, wiadomości, dzienników internetowych (tzw. blogów) itp. W wielu przypadkach komentarz może przesłać dowolna osoba nie będąca nawet zarejestrowanym i zalogowanym użytkownikiem danej witryny. Taka swoboda wypowiedzi jest często nadużywana.

Ludzie niejednokrotnie dają się ponieść emocjom i wysyłają komentarze wulgarne, obrażające inne osoby lub całkowicie niezgodne z tematem dyskusji. Niektórzy wykorzystują je również jako przestrzeń reklamową, podsyłając adresy stron WWW: własnych, produktów komercyjnych lub innych. Jak wykryć tego rodzaju niepoprawne komentarze, które niejednokrotnie szpecą nawet najbardziej poważną witrynę?

Obecnie prostą weryfikację dotyczącą istnienia adresów WWW, e-mail lub wulgaryzmów dokonuje się najczęściej automatycznie, gdyż wprowadzenie odpowiednich algorytmów nie jest szczególnie trudne i nie pociąga za sobą dużych kosztów. Bardziej zaawansowanej weryfikacji dokonują ludzie — moderatorzy.

Pomysł prawie w pełni automatycznego systemu weryfikacji komentarzy zrodził się w trakcie prac nad serwisem *gry.pl* (obecnie nieistniejącym w dawnej postaci). Młodzi użytkownicy serwisu dotyczącego gier online bardzo często umieszczali bezsensowne komentarze. Wraz ze wzrostem popularności coraz więcej czasu zajmowała moderacja komentarzy (usuwanie niepoprawnych).

Dzięki automatycznej weryfikacji komentarze niepożądane nie pojawiałyby się na witrynie, natomiast moderator mógłby w krótszym czasie sprawdzić ewentualne komentarze nierozstrzygnięte (czyli takie, co do których algorytm nie miał pewności, że są poprawne lub błędne).

1.1.2. Ograniczenia sprzętowe i programowe

Większość prac dotyczących klasyfikacji tekstów, do której można zaliczyć automatyczną weryfikację komentarzy, skupia się na rozwiązaniach tworzonych w językach kompilowanych (takich jak C, C++, Java itp.) z zastosowaniem własnych sposobów przecho-

wywania bazy wiedzy (w plikach o własnym formacie binarnym lub też w plikach tekstowych). Ponieważ do tworzenia aplikacji internetowych wykorzystuje się inne języki (skryptowe, interpretowane), wymusza to skorzystanie z jednego z tych języków.

Jako język skryptowy zostanie użyty PHP w wersji 5.1, gdyż jest to obecnie najpopularniejszy język stosowany w aplikacjach internetowych (według [26] działa na nim 40% aplikacji internetowych o różnym stopniu złożoności i korzysta z niego 22 mln. zarejestrowanych domen).

Z bardzo podobnych powodów jako system bazodanowych do testów zostanie wykorzystana baza danych MySQL w wersji 5. MySQL jest szybką bazą danych o wielu dodatkowych funkcjach. Liczba jej zastosowań w aplikacjach internetowych jest podobna do liczby wdrożeń stosujących PHP.

Ponieważ wiele witryn korzysta ze współdzielonych serwerów HTTP i dysponuje ograniczonymi zasobami pamięci operacyjnej oraz miejsca na dysku twardym, implementacja systemu weryfikacji powinna być możliwe zwięzła w kwestii przechowania danych.

System weryfikujący powinien działać możliwe jak najszybciej, by nie stanowił zbyt dużego obciążenia dla serwera a sami użytkownicy nie musieli zbyt długo oczekiwać na zakończenie przyjmowania wysłanego komentarza.

1.1.3. Podstawowe założenia dla systemu weryfikacji

Celem niniejszej pracy jest zbadanie możliwości odciążenia ludzi zajmujących się weryfikacją poprawności komentarzy przez implementację i wdrożenie kilku algorytmów automatycznego sprawdzenia komentarzy na podstawie budowanej inkrementacyjnie bazy wiedzy o tekstach poprawnych i niepoprawnych na danej witrynie. Istotne jest porównanie wyników uzyskiwanych przez poszczególne algorytmy a także jakość klasyfikacji najlepszego rozwiązania. Innymi słowy praca ma odpowiedzieć na pytanie, czy taka automatyzacja jest możliwa i na ile będzie skuteczna dla języka polskiego.

Implementacja algorytmów wymaganych do przeprowadzenia badań jest nowatorska, gdyż dotyczy weryfikacji komentarzy w języku polskim przy wykorzystaniu popularnego języka skryptowego i przy zastosowaniu serwera relacyjnej bazy danych. Wynika to z chęci sprawdzenia systemów klasyfikacji w najpopularniejszej konfiguracji serwerowej opisanej w poprzednim podrozdziale.

Praca zakłada, iż istnieje w miarę jednoznaczny sposób rozdzielania komentarzy na poprawne i niepoprawne. Przykładowo, za komentarze poprawne uznaje się jedynie te,

które dotyczą ściśle określonej dziedziny tematycznej. Bardziej ogólne witryny zawierające artykuły na wiele różnych tematów mają znacznie mniejszą szansę skorzystać z automatycznej weryfikacji, ponieważ grupy komentarzy poprawnych nie różnią się w wystarczającym stopniu od komentarzy niepoprawnych.

Wynika stąd, że system tworzenia automatycznej weryfikacji kierowany jest do administratorów witryn chcących posiadać komentarze lub dyskusje na ściśle określony temat (lub nawet otrzymywać artykuły o ściśle określonej tematyce). Dokonajmy jako przykład analizy witryny zapewniającej komentowanie umieszczonych na niej gier. Administrator witryny chce, by komentarze były związane tylko i wyłącznie z subiektywną oceną danej gry przez użytkownika. Komentarz nie powinien zawierać wyników rozgrywek, wymiany informacji między użytkownikami, innych danych nie związanych z recenzją itp.

Tworzony w niniejszej pracy system ma umożliwić początkowe nauczenie systemu weryfikującego niewielką liczbą komentarzy (do 50) uznawanych za poprawne i niepoprawne. System powinien uzupełniać swoją bazę wiedzy na podstawie nowych komentarzy zbieranych od użytkowników, o ile klasyfikacja danego komentarza nie była poprawna.

Aby zwiększyć skuteczność weryfikacji w przypadku niepewnych komentarzy (system nie potrafi jednoznacznie stwierdzić, czy jest to komentarz pozytywny, czy negatywny), tego rodzaju teksty nie powinny być automatycznie odrzucane ani zatwierdzane, ale kierowane do sprawdzenia przez moderatora w późniejszym okresie.

W takim podejściu moderator otrzymuje do sprawdzenia niewielką liczbę komentarzy, które mogłyby zostać niepoprawnie zaklasyfikowane.

1.2. ZAWARTOŚĆ KOLEJNYCH ROZDZIAŁÓW

Rozdział 2. stanowi wprowadzenie do stosowanych obecnie prostych systemów weryfikacji komentarzy. Dodatkowo szczegółowo opisuje dane testowe wykorzystywane w kolejnych rozdziałach.

Rozdział 3. zajmuje się dokładnym opisem kolejnych etapów przygotowywania tekstu do postaci odpowiedniej dla klasyfikatora. Omawia proces wychwytywania adresów e-mail i URL, tokenizacji, korekty ortograficznej, sprowadzania wyrazu do postaci słownikowej i wykrywania wulgaryzmów. Na końcu przedstawia wyniki testów wydajnościowych poszczególnych propozycji rozwiązań.

Rozdział 4. omawia sposoby mierzenia jakości klasyfikacji tekstów i definiuje samą klasyfikację. Zawiera pobieżny opis kilkunastu istniejących algorytmów klasyfikacji. Na końcu stara się porównać wyniki opisanych klasyfikatorów przedstawione w różnych pracach, by wybrać cztery rozwiązania implementowane w dalszej części pracy.

Rozdział 5. stanowi dokładny opis implementacji czterech wybranych algorytmów wraz z wyszczególnieniem ewentualnych zmian w nich prowadzanych. Dokładniej omawia czwarty algorytm klasyfikacji stanowiący propozycję autora pracy w kwestii rozwinięcia jednego z algorytmów (kNN) o możliwość uczenia inkrementacyjnego.

Rozdział 6. przedstawia miary jakości stosowane w testach oraz opisuje szczegółowo procedurę testowania. Prezentuje uzyskane wyniki i je omawia.

Rozdział 7. stanowi podsumowanie całej pracy. Wymienia zadania zrealizowane w pracy. Formułuje końcowe wnioski ze wskazaniem najlepszego algorytmu. Proponuje dodatkowo dalsze kierunki badań automatycznej klasyfikacji tekstów dla języka polskiego.

1.3. OBECNE ZASTOSOWANIA ANALIZY JĘZYKOWEJ I SYSTEMÓW KLASYFIKACJI

1.3.1. Analiza morfologiczna

Analiza morfologiczna tekstu stosowana jest obecnie w wielu dziedzinach informatyki związanych z przetwarzaniem wiedzy i rozpoznawaniem danych od użytkownika. Stanowi jeden z etapów w systemach rozpoznawania mowy, translacji tekstów (na języki obce lub język migowy), wydobywania znaczenia zdania i w wielu innych, w których potrzebna jest wiedza na temat szyku zdania, odmiany poszczególnych wyrazów a przede wszystkim ich rodzaju i lematu. Głównym zadaniem analizy morfologicznej jest próba ustalenia podstawowej formy sprawdzanego wyrazu oraz dostarczenie opisu jego odmiany (rodzaj, liczebność, możliwe znaczenie w zdaniu).

Dla języka polskiego istnieje wiele różnych systemów analizy morfologicznej (opisywanych między innymi w pracach [34] i [35]). Algorytmy kategoryzacji używają analizy językowej najczęściej w celu zmniejszenia liczby cech i tym samym poprawy szybkości oraz dokładności. Etap ten jest szczególnie ważny w obróbce języka polskiego z racji jego dużej fleksyjności, bo dla czasownika w formie bezokolicznika może istnieć nawet 100 odmian jak podaje [24].

Ponieważ algorytm weryfikacji powinien być możliwie szybki, w pracy skorzystano z systemu konwersji dowolnego wyrazu do jego wersji słownikowej (o ile wyraz jest zna-

ny) bez wydobywania innych informacji. Przyjęło się nazywać tego rodzaju system **lematyzatorem** [35]. W ten sposób dla języka polskiego uzyskuje się znaczące zmniejszenie ilości danych przy dużej szybkości.

1.3.2. Wykorzystanie systemów klasyfikacji tekstu

Systemy klasyfikacji tekstów w dużym stopniu korzystają z dwóch innych gałęzi informatyki: sztucznej inteligencji (AI — *Artifical Intelligence*) i eksploracji wiedzy (wydobywania informacji, IR — *Information Retrieval*). Ponieważ ilość danych tekstowych przechowywanych na cyfrowych nośnikach danych i dostępnych do pobrania w internecie stale się powiększa, rośnie również zapotrzebowanie na algorytmy klasyfikacji i filtracji dokumentów.

Liczba zastosowań kategoryzacji ciągle rośnie. Oto krótka lista kilku z nich [20]:

- filtracja nadchodzącej poczty elektronicznej, czyli eliminacja spamu;
- monitorowanie wiadomości, czyli informowanie o zdarzeniu najbardziej zainteresowanej tym osoby;
- ograniczanie strumienia napływających informacji jedynie do tych, którymi jesteśmy zainteresowani;
- przekierowanie poczty elektronicznej na podstawie jej zawartości do najbardziej kompetentnej osoby, która może ją obsłużyć;
- klasyfikacja napływających dokumentów (na przykład zgłoszeń patentowych) na podstawie zbioru predefiniowanych kategorii.

Szczególną uwagę warto zwrócić na eliminację spamu ([22], [32]), gdyż działanie tego rodzaju kategoryzacji jest bardzo podobne do weryfikacji komentarzy będącej tematem niniejszej pracy. Wynika to z ciągłego uczenia się algorytmu klasyfikacji na podstawie nowych rodzajów listów elektronicznych, którymi użytkownik nie jest zainteresowany.

Eliminacja spamu wykorzystująca znane od dawna algorytmy klasyfikacji (Naive Bayes, Winnow) osiąga doskonałą skuteczność wynoszącą w niektórych sytuacjach nawet powyżej 99% [32]. Z tego powodu jest implementowana w wielu klientach poczty i tym samym stanowi najczęściej wykorzystywany przez typowych użytkowników system klasyfikacji, choć nie mają oni o tym najmniejszego pojęcia.

1.4. RODZAJE DANYCH TESTOWYCH I SPOSÓB PROWADZENIA BADAŃ

1.4.1. Rodzaje danych testowych

Aby wyniki testów były w miarę zbliżone do wyników, które testowane klasyfikatory mogą osiągnąć w rzeczywistych warunkach, w pracy zastosowano rzeczywiste komentarze pochodzące z dwóch różnych portali oraz spreparowany zbiór danych testowych umożliwiający sprawdzenie systemu weryfikacji w przypadku danych o bardzo dobrej separacji.

Pierwszy zbiór danych testowych pochodzi z dawnej wersji serwisu *gry.pl* (obecnie witryna należy do innego właściciela) i zawiera 500 komentarzy sklasyfikowanych jako poprawne i 359 komentarzy sklasyfikowanych jako niepoprawne. Wszystkie komentarze zostały sklasyfikowane przez autora pracy. Komentarze mają związek z grami online i Java dostępnymi w serwisie.

Drugi zbiór danych testowych pochodzi z serwisu *gsmonline.pl* i zawiera 500 komentarzy sklasyfikowanych jako poprawne i 324 komentarzy sklasyfikowanych jako niepoprawne. Komentarze dotyczą własnych opinii na temat wiadomości ze świata telefonii komórkowej i stacjonarnej.

Trzeci zbiór komentarzy to zbiór sztuczny, który nie wystąpi w rzeczywistości, ale zapewnia oczywiste rozdzielenie dokumentów na poprawne i niepoprawne (przynajmniej przez człowieka). „Komentarze poprawne" to 500 przepisów kucharskich zebranych z witryny *http://www.przepisy-kulinarne.go-longhorn.net* natomiast komentarze niepoprawne to 375 komentarzy z serwisu *gsmonline.pl*.

Warto podkreślić, że dane nie są poddawane żadnej wcześniejszej obróbce przed ich wprowadzeniem do systemu. Innymi słowy, zawierają błędy ortograficzne, mogą być pozbawione znaków diakrytycznych, mają wiele znaków interpunkcyjnych, cyfry itp.

1.4.2. Sposób prowadzenia badań

Badania były przeprowadzane osobno dla każdego zbioru danych testowych. Aby w miarę wiernie zasymulować sposób korzystania z systemu w rzeczywistych warunkach, system jest początkowo uczony niewielką liczbą komentarzy poprawnych i niepoprawnych, które nie biorą udziału w późniejszych testach.

Już wstępne uczenie różni się od standardowych sposobów testowania jakości klasyfikatorów. W większości prowadzonych badań (na przykład w [18]) zbiór danych dzielony

jest na zbiór uczący i zbiór testowy, przy czym zbiór uczący jest często znacznie większy od zbioru testowego. W trakcie wykonywania testów baza wiedzy nie zmienia się. Podejście do prowadzenia testów dla komentarzy jest inne.

Po wstępnej nauce systemu weryfikacji komentarzy dochodzi do fazy testowej, w której system jednocześnie dokonuje klasyfikacji i uczy się na własnych błędach. System uaktualnia swój zbiór danych za każdym razem, gdy w trakcie prowadzenia testów popełni błąd (wynik automatycznej klasyfikacji komentarza jest porównywany z wykonaną wcześniej klasyfikacją ręczną) lub nie jest w stanie dokonać jednoznacznej klasyfikacji (podobieństwo komentarza do grupy pozytywnej i negatywnej jest na prawie jednakowym poziomie).

Takie badanie nie odwzorowuje dokładnie sytuacji występującej na witrynie, ponieważ moderator dokonuje analizy komentarzy tylko co pewien czas. Mimo to zastosowany inkrementacyjny test lepiej oddaje rzeczywiste warunki niż zastosowanie fazy uczenia z dużą liczbą komentarzy i fazy testowej bez możliwości aktualizacji zbioru danych.

1.5. ETAPY PRZETWARZANIA, TESTOWANE ALGORYTMY I UZYSKANE WYNIKI

1.5.1. Etapy przetwarzania tekstów

Ponieważ otrzymywany do klasyfikacji tekst jest dokładnie w takiej postaci, w jakiej wpisał go użytkownik, należy wykonać kilka kroków dokonujących konwersji komentarza na zbiór cech (tablicy wyrazów wraz z częstością ich wystąpienia) wymagany przez algorytmy klasyfikacji. Dodatkowo etap ten przeprowadza kilka testów pozwalających odrzucić komentarze niepoprawne bez dokonywania dalszej analizy.

Kolejne etapy przygotowania tekstu do kategoryzacji są następujące:

- sprawdzenie wystąpienia adresów e-mail i URL (jeśli są, komentarz uznaje się za niepoprawny, gdyż w wielu sytuacjach dotyczą stron użytkowników lub reklamy konkurencji);
- tokenizacja tekstu (rozdzielenie treści na wyrazy, zamiana liczb na wyrazy oznaczające przedziały wartości, pozbycie się znaków interpunkcyjnych, usunięcie wyrazów nie przynoszących żadnego pożytku w trakcie kategoryzacji);
- korekta ortograficzna wyrazów (uzyskane wyrazy w wielu sytuacjach nie są poprawne, więc warto spróbować uzupełnić w nich znaki diakrytyczne i odnaleźć poprawne formy);

- stemming, czyli redukcja do rdzenia będąca zamianą wyrazów na ich formy bazowe (w ten sposób zmniejsza się złożoność kategoryzacji i zwiększa jej skuteczność);
- testowanie wulgaryzmów (jeśli w przetworzonej liście wyrazów znajduje się wulgaryzm, komentarz należy odrzucić).

1.5.2. Zastosowane algorytmy klasyfikacji

Po przygotowaniu tablic wyrazów z częstością ich występowania dochodzi do właściwej klasyfikacji. Komentarze zostały sprawdzone w czterech algorytmach klasyfikacji zaimplementowanych od podstaw z wykorzystaniem PHP 5 i MySQL. Oto krótka charakterystyka wykorzystanych algorytmów:

1. Algorytm Balanced Winnow ([20], [32]) to bardzo prosty klasyfikator zapewniający łatwość inkrementacyjnego uczenia, gdyż wykorzystuje jednowarstwową sieć neuronową. Aby uzyskać zwiększenie szybkości nauki i jakości klasyfikacji, korzysta z dwóch wag dla każdego wyrazu. Algorytm doskonale nadaje się do implementacji z wykorzystaniem bazy danych. Aby zwiększyć szybkość jego działania, został wprowadzony mechanizm usuwania wyrazów, które w bardzo niewielkim stopniu uczestniczą w klasyfikacji. Zapewnia to czyszczenie zbioru wiedzy i jego dynamiczne dostosowywanie się.
2. Algorytm DCM+ to inkrementacyjna wersja algorytmu Discriminative Category Matching [13]. Podstawowa wersja algorytmu korzysta z rozdzielonych faz uczenia i klasyfikacji. Sam sposób zbierania informacji dopuszcza jednak modyfikację zbioru danych po nadejściu nowego dokumentu. Fakt ten został wykorzystany w trakcie implementacji. Dokonano również pewnych optymalizacji ze względu na wykorzystywanie jedynie dwóch kategorii komentarzy (oryginalny algorytm dopuszcza obsługę dowolnej liczby kategorii).
3. Algorytm k-NN ([4], [30]) jest wyjątkowo prosty w implementacji, gdyż nie korzysta z fazy uczenia i wszystkie obliczenia wykonuje w fazie kategoryzacji. Znajduje on k najbliższych sąsiadów (z dokumentów znajdujących się w bazie wiedzy) dla analizowanego komentarza. Przypisuje komentarz do tej grupy, która w k miała najwięcej reprezentantów. Niniejsza praca wprowadza kilka modyfikacji podstawowej wersji algorytmu: znaczenie ma nie suma reprezentantów, ale suma podobieństw do najbliższych reprezentantów danej grupy; stosowana jest

miara podobieństwa zapożyczona z DCM. Dodatkowo z bazy wiedzy po przekroczeniu określonego limitu dokumentów usuwane są te, które najdłużej nie brały udziału w żadnej kategoryzacji (zgodnie z regułami LRU).

4. Algorytm ICNN to nowa propozycja autora pracy starająca się wykorzystać zmiany wprowadzone w opisywanym powyżej algorytmie k-NN przy jednoczesnej próbie zastosowania łączonych podgrup dokumentów w sposób opisany w [14]. Rozwiązanie przedstawione w [14] (algorytm kNN, bez łącznika) nie jest inkrementacyjne, więc na potrzeby niniejszej pracy został opracowany algorytm dynamicznego scalania grup dokumentów lub aktualizacji najbliższej grupy o nowy dokument. Nowa propozycja okazała się najbardziej złożonym (w sensie długości kodu i liczby faz obróbki) algorytmem ze wszystkich czterech testowanych klasyfikatorów.

1.5.3. Krótkie omówienie wyników

Cztery algorytmy klasyfikacji tekstów wspomniane w poprzednim podrozdziale poddano kilku testom związanym z jakością kategoryzacji i szybkością działania. Osobno sprawdzono szybkość różnych podejść do przygotowywania tekstów do klasyfikacji.

Dwa z algorytmów (Balanced Winnow i DCM+) okazały się znacznie szybsze w działaniu od pozostałych dwóch. Jakość wszystkich czterech algorytmów przy wyłączonym progu niepewności odmawiającym kategoryzacji, gdy wartości oceny pozytywnej i negatywnej okazała się w miarę podobna. Poza trzecim sztucznie utworzonym zestawem komentarzy, w pozostałych dwóch uzyskane wyniki nie zachwyciły. Najlepszy z algorytmów nie potrafił przy najlepiej dobranym progu niepewności uzyskać średniego błędu łącznego w kilku testach poniżej 10% dla pierwszego zestawu komentarzy i 17% dla drugiego. Możliwe przyczyny takiego stanu rzeczy zostały opisane w podrozdziale 6.3.6.

W trakcie testów algorytmów przygotowywania komentarzy okazało się, że należy w miarę możliwości wystrzegać się korzystania z kodu pisanego w języku PHP na rzecz funkcji PHP (pisanych w C). Kod PHP jest znacznie wolniejszy od identycznego kodu napisanego w C. Dochodzi do sytuacji, w której teoretycznie znacznie szybszy algorytm napisany w PHP okazuje się wolniejszy od innej teoretycznie wolniejszej implementacji w C. Szczegółowe wyniki znajdują się w podrozdziale 3.6. Ze względu na wydajność zastosowano słownik Aspell i zwykłe wyszukiwanie w tablicach. Jedynie algorytmy niedostępne jako funkcje wykonano w PHP i wykorzystano w ostatecznym rozwiązaniu.

2. SPOSOBY MODERACJI KOMENTARZY I DANE TESTOWE

2.1. DOWOLNE KOMENTARZE NA WITRYNACH — PROBLEM DLA MODERATORÓW

Rozwój witryn internetowych i forów dyskusyjnych powoduje, że moderowanie komentarzy przesyłanych przez użytkowników staje się ogromnym wyzwaniem. Komentarze występują zarówno na dużych portalach, jak i na osobistych stronach WWW.

Możliwość umieszczenia w internecie dowolnej wypowiedzi jest przez użytkowników witryn bardzo często nadużywana. Im więcej nowych osób korzysta z sieci, tym więcej komentarzy wulgarnych lub obraźliwych. Nie wszyscy chcą stosować się do zasad netykiety.

Niektóre witryny nie są zainteresowanem ograniczaniem swobody wypowiedzi poszczególnych użytkowników i informują jedynie, że nie ponoszą odpowiedzialności za wypowiedzi poszczególnych osób. To rozwiązanie chroni twórców witryny od strony prawnej, ale nie zawsze przysparza nowych odwiedzających. Wyjątkiem mogą być witryny chcące zyskiwać popularność na coraz to wymyślniejszych obelgach. Stanowią one jednak tylko margines wszystkich stron WWW.

Obraźliwość czy wulgaryzmy to nie jedyne aspekty zmniejszające atrakcyjność witryn z komentarzami. Niejednokrotnie zdarza się, iż wbrew intencjom autora witryny związanej z konkretną tematyką, osoby umieszczają na niej komentarze o treści całkowicie odmiennej od pożądanej. Autor witryny i współpracujące z nim osoby muszą poświęcać dużo czasu na czytanie wszystkich komentarzy i usuwanie tekstów o nieodpowiedniej treści. Z racji dużego nakładu pracy wiąże się to ze znacznymi kosztami.

Istnieją dwa podstawowe podejścia do zgłaszania komentarzy lub wypowiedzi na forum. Pierwsze polega na automatycznym wyświetlaniu treści każdego nadesłanego komentarza. Moderator witryny okresowo usuwa niepoprawne komentarze. Przez pewien czas niepoprawny komentarz jest wyświetlany i niejednokrotnie na zasadzie lawiny wywołuje kolejne komentarze odnoszące się do niego (na przykład krytykujące autora wypowiedzi).

Drugie podejście powoduje zbieranie nadsyłanych komentarzy bez ich wyświetlania. Dopiero po sprawdzeniu treści przez moderatora pojawia się ona na witrynie. W tym rozwiązaniu niepoprawny komentarz co prawda nigdy nie zostanie umieszczony na witrynie, ale osoba go zgłaszająca musi czekać na jego pojawienie. W ten sposób ewentualne dysku-

sje w trybie „na żywo” są praktycznie niemożliwe, co często zniechęca osoby do komentowania.

Niezależnie od wybranego podejścia moderator musi poświęcić komentarzom czas proporcjonalny do popularności witryny i chęci komentowania zdarzeń przez użytkowników. Zwiększenie czasu poświęcanego na weryfikację komentarzy to zwiększenie kosztów działalności witryny. Poszukuje się rozwiązań, dzięki którym zmniejszy się nakład pracy moderatorów.

Niniejsza praca bada możliwość wykorzystania kategoryzacji tekstów do automatycznego odrzucania niepoprawnych komentarzy, stosując specjalne inkrementacyjne wersje algorytmów klasyfikacji, co zapewnia stałe uczenie się systemu na podstawie coraz to nowych danych.

2.2. OBECNIE STOSOWANE SYSTEMY WERYFIKACJI KOMENTARZY I ROZWIĄZANIE IDEALNE

Klasyfikacja tekstów, a co za tym stosująca ją automatyczna weryfikacja komentarzy, nie jest najprostszym rozwiązaniem starającym się ulżyć moderatorom. Kolejne podrozdziały zawierają informacje o kilku prostych systemach, ich wadach i zaletach. Ostatni z podrozdziałów informuje o podejściu idealnym.

2.2.1. Proste wyszukiwanie niedozwolonych sekwencji

Najprostszym podejściem do odrzucania nieodpowiednich komentarzy jest stworzenie listy wyrazów, zwrotów lub po prostu ciągów znaków, których pojawienie się automatycznie informuje, że dany tekst nie jest pożądany. Moderator okresowo uaktualnia listę niepożądanych zwrotów.

Podejście to zapewnia dużą szybkość i na ogół sprawdza się, gdy chcemy pozbyć się bardzo konkretnych stwierdzeń — wulgaryzmów, nazw konkurencyjnych witryn lub produktów itp. Niestety to rozwiązanie bywa niejednokrotnie łatwo oszukiwane przez użytkowników. Wystarczy dokonać prostego przekręcenia wyrazu, by wyraz nie został odnaleziony i komentarz go zawierający został wyświetlony. Przypuśćmy, że twórca witryny nie chce, by w komentarzach pojawiła się nazwa konkurencyjnej strony *totu.pl*. Komentarz zawierający taki tekst zostaje automatycznie odrzucony. Proste wyszukiwanie nie wychwyci jednak sytuacji, w której użytkownik wpisze: *to...tu.pl (usuń wielokropek)*. Taką sytuację wykryje dopiero moderator przeglądający nadesłane komentarze. Dodatkowo pro-

ste wyszukiwanie wyrazów wulgarnych może spowodować odrzucenie komentarza z poprawnymi wyrazami, na przykład z powodu zastosowania wyrazu abstra**huj**ąc.

2.2.2. Złożone wyszukiwanie niedozwolonych sekwencji

Kolejne rozwiązanie stanowi rozwinięcie poprzedniego. Zamiast prostego wyszukiwania stwierdzeń znajdujących się na liście zabronionych wyrazów, stosuje się wyrażenia regularne. Dzięki nim możliwe jest bardziej rozmyte określenie odrzucanych stwierdzeń. Użytkownicy chcący obejść prostą blokadę zapiszą wulgarny wyraz *chuj* jako *hhhuuujjjjj*. Nawet wypisanie wszystkich podstawowych kombinacji wyrazu w prostym wyszukiwaniu (*chuj*, *huj*, *chój*, *hój*) nie spowoduje wyłapania takiej sytuacji, ale wyrażenie regularne w postaci `\b(c?)+h+(u|ó)+j+\b` radzi sobie z wszystkimi przypadkami (także powtórzeniami wyrazów).

Aby skorzystać z tego bardziej zaawansowanego podejścia, osoba zajmująca się moderacją komentarzy musi znać i rozumieć wyrażenia regularne. Kolejną wadą tego rozwiązania względem poprzedniego jest większa złożoność obliczeniowa. Kompilacja i sprawdzanie wyrażeń regularnych zajmuje więcej czasu niż proste przeszukanie.

2.2.3. Użytkownicy oceniają użytkowników

Wiele portali wprowadziło pomysłowe rozwiązanie, w którym to użytkownicy danej witryny zgłaszają moderatorom komentarze do skasowania. Przy każdym komentarzu pojawia się specjalne łącze pozwalające poinformować prowadzącego witrynę, by skasował wskazany komentarz. Moderator może, ale nie musi skasować komentarza wskazanego przez odwiedzającego.

To rozwiązanie ma kilka zalet. Jest bardzo proste do wprowadzenia, nie obciąża systemu dokonywaniem weryfikacji treści, moderatorzy są informowani najczęściej o komentarzach, które drażnią innych użytkowników, i mają znacznie mniej pracy, ponieważ zajmują się przede wszystkim zgłoszeniami bez potrzeby przeglądania wszystkich nowych wpisów. By skłóceni ze sobą użytkownicy nie donosili na siebie nawzajem, zgłaszając prośby o usunięcie komentarzy (co mogłoby znacznie zwiększyć liczbę spraw analizowanych przez moderatorów), wprowadza się próg zgłoszeń. Oznacza on, że dopiero określona liczba zgłoszeń usunięcia jednego komentarza od wielu różnych użytkowników powoduje przyjrzenie się sprawie przez moderatora.

Do wad takiego podejścia można zaliczyć pojawienie się niepoprawnych komentarzy na witrynie przynajmniej przez pewien okres czasu. Co ważne, czas ten może być tym dłuższy, im mniejsze jest zainteresowanie tematem, którego dotyczył nieodpowiedni komentarz. Użytkownicy niekonieczne muszą mieć to samo zdanie w kwestii komentarza (poprawny lub niepoprawny) co moderator witryny. W takiej sytuacji moderator może nie dowiedzieć się w łatwy sposób o komentarzu, który sam uznałby za niepoprawny, ale nie przeszkadza on użytkownikom. Przykładem takiej sytuacji jest opublikowanie w komentarzu logina i hasła innego użytkownika dającego większe uprawnienia do aktualnej strony. Osoby odwiedzające mogą nie być zainteresowane zgłoszeniem takiej sytuacji.

2.2.4. Rozwiązanie idealne

Rozwiązanie idealne dotyczące znacznego zmniejszenia nakładu pracy moderatora powinno cechować się następującymi właściwościami:

- nie dopuszczać do wyświetlenia komentarza, co do którego istnieje duże prawdopodobieństwo niepoprawności;
- nie wymagać od moderatora wiedzy technicznej (związanej na przykład z konstrukcją wyrażeń regularnych);
- jedynie komentarze o podejrzanej zawartości (przy braku silnego wskazania, czy jest on poprawny, czy błędny) kierować do moderatora w celu rozpatrzenia przed ich wyświetleniem;
- uczyć się wraz z nadchodzeniem nowych komentarzy, zapewniając tym samym ewentualny dryft linii podziału poprawny lub niepoprawny;
- nie stosować długiego i żmudnego trybu przygotowań wstępnych;
- nie obciążać w znaczącym stopniu serwera;
- w niemal natychmiastowy sposób informować w większości przypadków o przyjęciu lub odrzuceniu komentarza.

Z przedstawionego opisu cech systemu idealnego wynika, że powinien on uczyć się na przykładach komentarzy zatwierdzanych lub odrzucanych przez moderatora poprawnego podejmowania decyzji w sposób automatyczny. Skoro mowa o nauce przez przykłady i automatycznej ocenie, oznacza to potrzebę wykorzystania algorytmów kategoryzacji tekstów.

W niniejszej pracy rozważono, czy osiągnięcie przedstawionego ideału jest możliwe przy wykorzystaniu obecnej wiedzy na temat kategoryzacji tekstów i zastosowaniu najpo-

pularniejszych elementów składowych aplikacji internetowych — języka skryptowego PHP 5 i bazy danych MySQL 5.

Aby zapewnić możliwie najlepszą odpowiedź na to pytanie, zostaną wykorzystane rzeczywiste komentarze w języku polskim.

2.3. SZCZEGÓŁOWY OPIS DANYCH TESTOWYCH

2.3.1. Komentarze serwisu gry.pl

Pierwszy zbiór komentarzy używanych w trakcie testów pochodzi z nieistniejącej już wersji serwisu *gry.pl* (obecnie serwisem zarządza inna firma). Komentarze zostały zebrane na początku 2006 roku. Dotyczą one gier online prezentowanych w serwisie oraz sprzedawanych gier Java.

Warto podkreślić, iż brakuje w zebranym zbiorze części komentarzy najbardziej wulgarnych lub wyjątkowo złośliwych, gdyż zostały one usunięte wcześniej przez moderatorów serwisu. Nie istniała szansa zbierania przez pewien okres czasu wszystkich możliwych komentarzy, by uzyskać pełne spektrum nadsyłanych tekstów.

Komentarze zostały podzielone na poprawne i niepoprawne na potrzeby tej pracy przez jej autora z zastosowaniem następujących reguł:

- Komentarze poprawne to takie, które informują o subiektywnej ocenie gry, mogą być emocjonalne lub stronnicze.
- Komentarze niepoprawne to takie, które zawierają wyniki rozgrywek, opis działania gry, możliwe do zastosowania sztuczki lub całkowicie nie dotyczą tematu gier (nie mogą być potraktowane jako opinia).

Ze zbioru ponad 13 tysięcy komentarzy ręcznie sklasyfikowanych zostało 859 (500 pozytywnych i 359 negatywnych). Pozostałe komentarze zostały oznaczone jako niesklasyfikowane. Komentarze z serwisu są stosunkowo krótkie — ich długość najczęściej nie przekracza kilkunastu wyrazów a niejednokrotnie zdarzają się treści jednowyrazowe.

Tabela 2.1 przedstawia kilka przykładowych komentarzy sklasyfikowanych jako poprawne i błędne.

Tabela 2.1. Przykłady komentarzy testowych z serwisu gry.pl

Komentarz poprawny	Komentarz niepoprawny
zgadzam się z nelsonem! totalna katastrofa! i to ma być gra zręcznościowa? za takie gry ta ja dziękuje! nędza! po prostu nie da się w to grać!	mój rekord to 512365
ALE FAJNA GGGIIIEEERRRKKKAAA!!!	Konstantynopolitańczykówna hihihi:D
Może być ale jest za łatwo wygrać z komputerem	ale to trudne uuuu gg:3442272
beznadzieja...	lokijygrdwalmjngvdxaz
zgadzam sie z grodzia ta gra jest najlepsiejsz w całym serwisie!!!!!!!!!!!!!!!!!!1	adi twuj stary to frajer^^

Warto zwrócić uwagę na stosowanie wielkich liter w komentarzach, powtórzenia wyrazów, błędy ortograficzne i niestosowanie znaków diakrytycznych oraz wystąpienia wartości liczbowych (wyniki, numery GG).

2.3.2. Komentarze serwisu gsmonline.pl

Drugi zbiór komentarzy testowych pochodzi z serwisu *gsmonline.pl* [38] stanowiącego portal tematyczny dotyczący telefonii komórkowej i stacjonarnej. Komentarze zostały zebrane z kilku aktywnie komentowanych informacji prasowych w kwietniu 2006 roku.

Serwis wykorzystuje opisywany w podrozdziale 2.2.3 system wskazywania moderatorom komentarzy, które powinny zostać usunięte. Oznacza to, że brakuje w zebranym zbiorze co bardziej wulgarnych albo złośliwych komentarzy (choć są wyjątki, które z racji niewielkiej szkodliwości pozostały niezauważone przez obsługę portalu).

Komentarze z tego zbioru zostały podzielone na poprawne i niepoprawne na potrzeby niniejszej pracy przez jej autora z zastosowaniem następujących reguł:

- Jako komentarze poprawne zostały oznaczone te teksty, które mogły wnieść coś konkretnego do prowadzonych dyskusji na temat polskiej telefonii komórkowej i nie rozmijały się znacząco ze specyfiką portalu. Ocena ta była mocno subiektywna.
- Jako komentarze niepoprawne zostały uznane informacje mocniej odbiegające od tematyki portalu (w szczególności licytacje na mało znaczące argumenty), „wycieczki osobiste", opisywanie innych użytkowników (szczególnie obraźliwe) i konkurencji portalowej itp. W tej grupie znalazły się również komentarze z wulgaryzmami i adresami URL.

Niejednokrotnie moderator w przypadku komentarzy kieruje się własną subiektywną oceną wartości tekstu. Dwóch moderatorów tego samego serwisu nie zawsze musi w dokładnie taki sam sposób dopuszczać lub odrzucać komentarz, co podkreśla między innymi [31]. Drugi zbiór danych testowych jest obarczony największą subiektywnością podziału.

Ze zbioru ponad 1 600 komentarzy ręcznie sklasyfikowanych zostało 824 (500 pozytywnych i 324 negatywnych). Pozostałe pozostawiono jako niesklasyfikowane. Zadziwiające okazało się podobieństwo stosunków typów komentarzy obu zbiorów danych. Komentarze były analizowane po kolei aż do uzyskania 500 komentarzy pozytywnych. W obu sytuacjach stosunek komentarzy pozytywnych do negatywnych wynosi mniej więcej 3:2. Został on powtórzony również w trzecim, sztucznie utworzonym podziale (tym razem z premedytacją).

Komentarze mają bardzo różną długość — od kilku wyrazowych po nawet 70-cio wyrazowe. Witryna, na której się znalazły, ogranicza długość pojedynczego komentarza do 500 znaków. Duży odsetek komentarzy (około 30%) jest bliski tej granicy natomiast tekstów bardzo krótkich jest niewiele. Komentarze w wielu przypadkach są napisane bez polskich znaków diakrytycznych, ale poprawnie ortograficznie. Pojawia się również wiele nazw własnych lub skrótów.

Tabela 2.2 przedstawia kilka przykładowych komentarzy sklasyfikowanych jako poprawne i błędne.

W porównaniu z pierwszym zbiorem testów tutaj pojawiają się komentarze bardziej poprawnie językowo, z bogatszym słownictwem i stosunkowo niewielką liczbą znaków specjalnych, choć niejednokrotnie bez polskich znaków diakrytycznych. W niektórych sytuacjach komentarze niewiele się od siebie różniące trafiają do różnych kategorii. Stosunkowo łatwo tu o pomyłkę w klasyfikacji nawet przez eksperta.

Tabela 2.2. Przykłady komentarzy testowych z serwisu gsmonline.pl

Komentarz poprawny	Komentarz niepoprawny
obniza. to dobrze ze obniza. niech inni tez obnizaja. i przydaloby sie zmniejszenie limitow na pobieranie/wysylanie danych	Eeee... już nie będę się wyżywał ;-) mam inne rzeczy na głowie teraz ;-) Zajrzę później pod ten wątek.
za tydzień pewnie przedłużą bo w głębsze zmiany nie chce mi się wierzyć :)	zajebista obnizka... juz sie cieszylem jak zobaczylem newsa, w srodku to co zwykle podaje Orange.... buuuuuuuuuuu
Przepraszam - w innych uwarunkowaniach. Pakiety w Erze też mają termin , tlko zgoła inny i jakby bardzie sensowny i UCZCIWSZY. Jak się chce to można , jak widać.	issa - co miałeś na myśli?
Operator wprowadza ryczałt a potem "dziwi" się, że ludzie korzystają ;-) Ludzie decydowali się na tą ofertę bo była po prostu lepsza od oferty TP i należało oczekiwać dużego zainteresowania. A jeśli większe zainteresowanie to należy pomysleć o łączach i przepustowości. A jeśli klienci "narażają na duży deficyt" to kto robił analizę opłacalności takiego przedsięwzięcia? Proste związki przyczynowo-skutkowe a taki problem dla operatora.	*** TU KTOS INNY jest ograniczony:)
*lennox: całkowicie Cię popieram. ostatnio przy slynnej promocji Ery myślałem nad przeprowadzeniem pewnych bardziej złożonych transakcji kupna-sprzedaży i dokonałem pewnych kalkulacji. dla mnie to też jest frustrujące że operator więcej i to *znacznie* dopłaca do telefonów (rozumiane jako różnica między ceną w promocji, a ceną na allegro) dwóm klientom którzy będą płacić goły abonament po 25 złotych, niż jednemu który będzie przez 2 lata płacił po 75 złotych. gdzie logika?? :(((	z tym przełożonym to... takie Polskie ! przełożony - idiota, bo ma plecy, a wyniki jakieś tam są, bo pracownicy sobie muszą żyły wypruwać... *marzasz - wiesz co ? nie chce mi się podawać przykładów, podawałem je już na tym forum dziesiątki razy, a są to długie historie, generalnie to mam tu na myśli ideę, a potem orange który jest identyczny, to samo hamstwo, zlewka na klienta, brak profesjonalizmu, indywidualnego podejścia itp. a akcje jakie idea/orange robi poprostu mrożą krew w żyłach...

2.3.3. Przepisy kucharskie kontra komentarze serwisu gsmonline.pl

Trzeci zbiór danych testowych jest nietypowy. Jego podstawowym zadaniem jest sprawdzenie jakości klasyfikacji w przypadku wyjątkowo silnej separacji dwóch kategorii tekstów. Zbiór „komentarzy pozytywnych” to 500 rzeczywistych przepisów kucharskich zebranych z witryny *http://www.przepisy-kulinarne.go-longhorn.net*. Przepisy dłuższe niż 500 znaków zostały ograniczone do zadanego limitu długości. „Komentarze negatywne” to 375 komentarzy uznanych za poprawne z drugiego zbioru testowego.

Dla tak dobranych danych istnieje pewność co do eliminacji czynnika ludzkiego wpływającego na podział tekstów na kategorie. Każda osoba bez przeszkód i ze 100% skutecznością dokonałaby poprawnego podziału danych na dwie kategorie. Trzeci zbiór danych pozwala sprawdzić, czy automatyczna kategoryzacja tak dobrze jak człowiek poradzi sobie z tego rodzaju danymi.

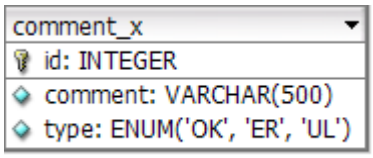
Tabela 2.3 przedstawia po dwa przykładowe przepisy kucharskie i komentarze.

Tabela 2.3. Przykładowe przepisy kucharskie i komentarze z trzeciego zbioru danych testowych

Komentarz poprawny (przepis)	**Komentarz niepoprawny (opinia)**
Banany zapiekane 4 szt. banan przeciąć na pół, wyjąć ze skórki, pokroić w plasterki5 łyżka wiśniówka 5 łyżka cukier imbir 3 łyżka masło rozpuścić Pokrojone banany wymieszać z wiśniówką, połową cukru i imbirem, odstawić na pół godziny. Napełnić masą skórki bananów ułożone w naczyniu do zapiekanek, polać masłem i posypać cukrem. Piec 20 min w 200°C.	Zdecydowanie, jak na modemy tylko gprs albo tylko umts to ceny zdecydowanie za wysokie i za długa umowa. Generalnie oferta do poprawki :)
Baranina po podlasku 70 dag baraniny, 10 dag wędzonej słoniny, 4 duże cebule, papryka,, pół litra bulionu, 3 pomidory, sól. Mięso pokroić w kostkę, a cebulę w plasterki. Słoninę stopić i szybko podsmażyć na niej baraninę. Dodać cebulę, paprykę i razem poddusić. Po 10 minutach wlać bulion, dodać pokrojone pomidory i dusić przez 1-1,5 godziny. Podawać z makaronem.	Acha :-) na koniec Plusa pochwale za ebok ! Super sprawa że można ładnie samemu sobie zmienić numer na życzenie :) w Erze strasznie mi to przeszkadza że wybór numerów można zrobić od ręki tylko w salonach firmowych a w Orange cenny za wybór numerów są kosmiczne :-)

2.3.4. Format przechowywania danych testowych

Dane testowe zostały umieszczone w osobnych tabelach bazy danych — po jednej tabeli na rodzaj danych testowych. Każda tabela składa się z trzech pól: identyfikatora liczbowego, treści komentarza i jego typu (patrz rysunek 2.1, gdzie x oznacza numer serii danych testowych). W typie wyliczeniowym (`type`) kod `OK` oznacza komentarz poprawny, `ER` niepoprawny natomiast `UL` nieskategoryzowany.



Rysunek 2.1. Format tabel przechowujących dane testowe

Treść komentarzy nie jest w żaden sposób modyfikowana i przygotowywana przed dostarczeniem do mechanizmu automatycznej weryfikacji.

3. ANALIZA JĘZYKOWA TEKSTU I JEGO PRZYGOTOWANIE DO KATEGORYZACJI

Niniejszy rozdział opisuje wszystkie etapy sprawdzania tekstu i jego konwersji do postaci najbardziej odpowiedniej dla klasyfikatorów tekstów — zbioru słów dokumentu wraz z częstością ich wystąpienia. W przypadku tekstów w języku polskim o słabej jakości (duża liczba błędów, nietypowa składnia) wstępne przygotowanie danych staje się poważnym wyzwaniem i w istotny sposób wpływa na jakość automatycznej weryfikacji.

Poszczególne podrozdziały zajmują się kolejnymi etapami przetwarzania tekstu. Ostatni podrozdział przedstawia wyniki uzyskane dla poszczególnych propozycji rozwiązań i motywuje wybór podejścia stosowanego dla wszystkich późniejszych algorytmów klasyfikacji.

3.1. WYRAŻENIA REGULARNE W WYCHWYTYWANIU ADRESÓW URL I E-MAIL

W automatycznej weryfikacji komentarzy założono, że automatycznie odrzucane mają być wszystkie komentarze zawierające adresy URL lub e-mail, gdyż najczęściej wskazują one strony konkurencji, stanowią reklamę (np. w celu zwiększenia pozycji własnej witryny w wyszukiwarce internetowej) lub dotyczą spraw osobistych.

3.1.1. Adresy URL

Wykrycie adresu URL wbrew pozorom nie jest sprawą prostą. Wydawałoby się, że wystarczy zastosować powszechnie znane techniki wykrywania tego rodzaju konstrukcji. Dokonują jej komunikatory internetowe i klienty poczty elektronicznej w celu utworzenia z adresu interaktywnego łącza, stosując dla całego tekstu wyrażenia regularne dopasowujące się tylko i wyłącznie do adresów URL w pełni zgodnych ze standardem. Niestety, gdy tylko użytkownik chcący podesłać komentarz zawierający adres URL zorientuje się, że istnieje standardowa blokada, dokona rozbicia adresu w taki sposób, by był on czytelny i jednoznaczny dla człowieka a jednocześnie nie był wykrywany przez standardowe wyrażenie regularne.

Prześledźmy to na przykładzie. Użytkownik chce zareklamować własną witrynę *www.totu.pl*, ale adres taki wychwytuje standardowy mechanizm weryfikacji. Przy kolejnej próbie postara się w inny sposób wskazać adres, stosując na przykład znak + zamiast znaku kropki do rozdzielania członów adresu i usuwając opcjonalny w większości sytuacji

człon www — *totu+pl*. Tego rodzaju złączenie nie zostaje wykryte przez standardowe mechanizmy poszukiwania adresu URL.

Wyrażenie regularne [12] stosowane do analizy tekstu pod kątem adresów URL powinno więc uwzględniać tego rodzaju sytuacje. Pojawia się jednak inny problem. Zbyt luźne dopasowywanie się (czyli zbyt agresywne i podejrzliwe wyszukiwanie) potrafi w wielu przypadkach błędnie wykryć adres. Twórca wyrażenia regularnego mógłby pokusić się o uznawanie za próbę wpisania adresu URL tekstu *totu pl. św. Piotra* (traktowanie spacji jako ewentualnego separatora) W tym przypadku jednak autor komentarza nie chciał reklamować strony, ale omyłkowo zabrakło spacji między *to* i *tu* w prawdopodobnie poprawnym komentarzu.

Aby zapewnić wykrywanie większej liczby adresów URL niż standardowe rozwiązania i jednocześnie uniknąć omyłkowego traktowania jako adresów URL poprawnych tekstów, w niniejszej pracy użyte zostało następujące wyrażenie:

```
[a-zA-Z][\w]*([\.+][\w]+\*)*([\.+\*]↵
(pl|com|org|net|edu|info|biz|fm|eu|de|fr|uk))(\W|$)
```

Znak ↵ oznacza, że przejście do następnego wiersza następuje tylko w tekście pracy (w rzeczywistości wyrażenie pisane jest jako jeden ciąg). Warto zauważyć, że w celu ograniczenia pomyłek kosztem mniejszej skuteczności nie stosuje typowego podejścia do końcówek adresowych (na przykład `\w{2,4}`).

3.1.2. Adresy e-mail

Adresy e-mail należy potraktować podobnie, jak adresy URL, czyli wykrywać nie tylko najbardziej oczywiste przypadki. Z drugiej strony również tu nie warto podejrzewać każdego możliwego fragmentu, by nie zwiększyć liczby błędnych dopasowań.

Przedstawione poniżej wyrażenie regularne wykrywa nie tylko standardową sytuację, czyli wystąpienie znaku `@` pomiędzy wyrazem i dwoma wyrazami z kropką pomiędzy (*tadek@test.pl*), ale również stosowane często rozwiązania alternatywne (*tadek at test.pl* lub *tadek(at)test.pl*). Oczywiście nawet w tym podejściu użytkownik może łatwo oszukać wyrażenie regularne, stosując bardziej wyrafinowane modyfikacje.

```
\w[-._\w]*\w.?(@|[( ]at[) ])\w[-._\w]*\w\.\w{2,4}
```

Sprawdzanie adresów URL i e-mail to jedyne operacje wykorzystujące wyrażenia regularne w całej automatycznej weryfikacji. Jeśli w komentarzu adres nie zostanie odnaleziony, można przejść do kolejnej fazy przetwarzania, czyli rozbijania tekstu na wyrazy.

3.2. TOKENIZACJA, ZAMIANA WARTOŚCI LICZBOWYCH I STOPLISTA

Tokenizacja, czyli podział tekstu na elementy składowe zwane tokenami, zapewnia wydobycie listy wyrazów z tekstu. Na tym etapie usuwane są wszelkiego rodzaju znaki przestankowe, gdyż nie wpływają one w żaden sposób na późniejszą kategoryzację. Lista wyrazów po tokenizacji wraz z ich liczebnością to tak zwany BOW (ang. *bag of words*, czyli worek słów) [6].

Większość systemów kategoryzacji usuwa na tym etapie liczby. W tej pracy zastosowano jednak inne podejście. Odnalezione wartości liczbowe są w prosty sposób zamieniane na kilka słów informujących o zakresie wartości. Przykładowo, wartość 27 zostaje zamieniona na słowo „dziesiątka" natomiast wartość 1342 na słowo „tysiąc". Unikanie odrzucania liczb może przyczynić się do poprawy jakości automatycznej weryfikacji komentarzy. Wystarczy przyjrzeć się przykładowym komentarzom: zawierają one wyniki rozgrywek, modele telefonów i proporcje wagowe. Choć system zamiany jest wyjątkowo prosty, zapewnia lepsze rozróżnienie komentarzy, gdy moderator chce, by liczby pojawiały się tylko w komentarzach poprawnych lub tylko błędnych.

Kolejną nowością względem tradycyjnych systemów przygotowujących BOW jest pozbywanie się powtórzeń znaków z wydobywanych wyrazów. Modyfikację tę wymuszają sami użytkownicy, gdyż niejednokrotnie wpisują wyrazy w postaci typu *tttaaakkk*, *SSSUUUPER* itp. W języku polskim stosunkowo niewiele poprawnych ortograficznie wyrazów posiada powtórzenie liter występujące obok siebie, więc można z dużą dozą prawdopodobieństwa założyć, że powtórzenia są celowym działaniem użytkowników i należy się ich pozbyć. Nawet jeżeli wyraz miał zawierać powtórzenie, z łatwością zostanie ono przywrócone przez system korekty ortograficznej, o ile tylko powstały wyraz nie będzie poprawny (na przykład wyraz *mięki* korekta poprawi bez problemów na *miękki*). Niestety są sytuacje (choć na szczęście niezwykle rzadkie), w których usuniecie powtórzenia prowadzi do powstanie całkowicie innego, poprawnego wyrazu (na przykład *lekki* zmienia się w *leki*). Błędowi tego typu można zapobiec, tworząc słownik z takimi przypadkami. Nie zostało to jednak wprowadzone w testowanym systemie.

Zamiana wielkich liter na małe stanowi niemalże standardowy element każdego systemu tokenizacji przygotowującego dane dla klasyfikatora. W przedstawianej pracy również została wprowadzona.

Aby zmniejszyć obciążenie dalszych etapów analizy i kategoryzacji mało znaczącymi wyrazami oraz poprawić jakość klasyfikacji, wiele systemów stosuje tak zwaną stopli-

stę [21]. Stoplista to zbiór wyrazów usuwanych z BOW. Jej główne zadanie polega na pozbywaniu się spójników, zaimków i innych części mowy w żaden sposób nie wpływających na wybór kategorii. W stopliście warto również umieścić wyrazy występujące w języku niezwykle często, czyli wyrazy o niewielkiej sile rozróżniającej.

Na potrzeby niniejszej pracy została wykonana stoplista zawierająca 161 pozycji. Stanowi kombinację wyrazów zebranych przez autora pracy i najczęstszych wyrazów występujących w języku polskim (na podstawie słownika frekwencyjnego demonstracyjnej wersji korpusu PWN, [28]). Lista wyrazów została ułożona w ten sposób, by wyrazy o największym prawdopodobieństwie wystąpienia znalazły się jak najwyżej. Ponieważ korzystanie ze stoplisty ma miejsce przed korektą ortograficzną, lista wyrazów zawiera również najczęściej występujące błędne wersje wyrazów. Dodatkowo system zawsze pomija wyrazy jedno- i dwuliterowe (w języku polskim są to najczęściej spójniki i zaimki).

Tokenizer dodatkowo zlicza sekwencje co najmniej czterech znaków gwiazdki, co w większości sytuacji świadczy o zastąpieniu wulgarnego wyrazu jego łagodną odmianą (na przykład *napierdolić* jest zamienione na *napie****ić*). Informacja o takich wystąpieniach może zostać wykorzystana na dalszych etapach analizy.

3.3. KOREKTA ORTOGRAFICZNA

3.3.1. Ogólna koncepcja

Przyglądając się przykładowym komentarzom prezentowanym w poprzednim rozdziale, można zauważyć, iż w wielu przypadkach wyrazy nie są zapisywany w sposób poprawny ortograficznie. Gdyby po tokenizacji pozostawić je w oryginalnej postaci, znacząco ucierpiałaby na tym jakość klasyfikacji. Wynikałoby to z istnienia wielu wersji tego samego wyrazu w różnych postaciach. Każda z postaci byłaby przez algorytmy klasyfikujące traktowana niezależnie.

Warto w tym miejscu podkreślić, że większość systemów klasyfikacji sprawdza się na standardowych zbiorach testowych, które są pozbawione błędów ortograficznych i celowych przekłamań wprowadzanych przez twórców. Aby zapewnić jak najlepsze odtworzenie poprawnej wersji wyrazu (o ile to możliwe), warto przed zwykłą korektą ortograficzną sprawdzić, czy błędny wyraz nie stanie się poprawnym słowem po uzupełnieniu diakrytyki. Istnieje wtedy znacznie mniej kombinacji, co oznacza, że operacja ta jest znacznie szybsza i dokładniejsza od pełnej próby poprawy ortografii.

Korektę ortograficzną w języku PHP można przeprowadzić na kilka sposobów. Istnieje moduł `pspell` zapewniający dostęp do programu i biblioteki Aspell stanowiącej standardowy system sprawdzania pisowni wykorzystywany w systemach Linux. Drugie rozwiązanie to lista poprawnych wyrazów przechowywana w bazie danych w postaci tabeli z założonym odpowiednim indeksem. Trzecie podejście polega na napisaniu własnego algorytmu korekty ortograficznej bezpośrednio w języku PHP.

W niniejszej pracy zostało sprawdzone rozwiązanie pierwsze i trzecie.

3.3.2. Aspell

Program i biblioteka Aspell [39] pozwala w sposób wydajny nie tylko dokonywać sprawdzenia, czy wskazany wyraz znajduje się w słowniku ortograficznym (czy jest poprawny), ale także przedstawiać sugestie dotyczące jego poprawy. Niestety w testowanej wersji programu Aspell — 0.50.3 — nie jest dostępny przełącznik zapewniający dokonywanie jedynie korekty znaków diakrytycznych.

Język PHP 5 zawiera opcjonalny moduł `pspell` potrafiący skorzystać z programu Aspell. Moduł udostępnia funkcje wyboru słownika, określania dodatkowych opcji korekty, sprawdzania, czy wyraz jest w słowniku, oraz sugerowania poprawnych wersji wyrazu po przekazaniu błędnego słowa.

Autorzy programu Aspell testowali jedną z jego wcześniejszych wersji (0.30) w różnych trybach pracy i porównywali ją z programami Ispell oraz Word 97. Test był wykonywany z wykorzystaniem wyrazów i słownika dla języka angielskiego. Aspell w wersji najszybszej (`FAST`) proponował w trakcie korekty poprawny wyraz w pierwszych 5 zwracanych wynikach w 81% przypadków. W pierwszych 25 zwracanych wynikach poprawna wersja wyrazu znajdowała się w 87% przypadków. Aspell w wersji podstawowej (`NORMAL`) okazywał się lepszy o kilka procent, ale czas jego działania był kilkakrotnie dłuższy (dokładne testy szybkości zostały przedstawione w [8]).

Program dokonujący sprawdzania i korekty danych to tylko jeden z elementów układanki. Równie istotny jest sam słownik. Jako podstawowy słownik ortograficzny w niniejszej pracy został wykorzystany alternatywny słownik ortograficzny języka polskiego [3]. Ponieważ istnieje między innymi w wersji specjalnie przygotowanej dla programu Aspell 0.50, został on użyty bez modyfikacji. Słownik zawiera ponad 3,2 miliona wyrazów i po instalacji w systemie Aspell zajmuje na dysku 62 MB.

3.3.3. Korekta ortograficzna zaimplementowana bezpośrednio w PHP

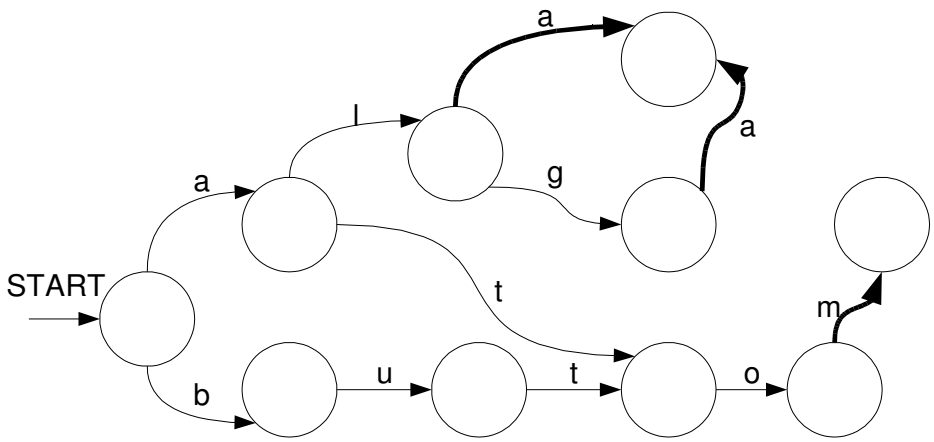
W jaki sposób stosując tylko język PHP zapewnić wydajne przeszukiwanie słownika ortograficznego zawierającego ponad 3 miliony wyrazów o różnej długości? Z pomocą przychodzą deterministyczne automaty skończone. Dzięki nim możliwa jest kompresja wszystkich wyrazów słownika (lista wyrazów zajmuje 40 MB) do 1,2 MB. Co ciekawe, nie jest to jeszcze postać minimalna, do której można zmniejszyć słownik z 3 milionami wyrazów. Zakodowany plik danych automatu skończonego nadaje się do dodatkowej zwykłej kompresji, po której powstaje pliki o rozmiarze około 0,7 MB.

Sposób tworzenia automatu skończonego kodującego słownik został opisany w pracach [7], [9]. Na potrzeby niniejszej pracy nie było potrzebne tworzenie kodu programu, który konwertowałby listę słów do acyklicznego, deterministycznego automatu skończonego, ponieważ są dostępne tego rodzaju programy. Skorzystano z biblioteki i programów udostępnianych przez pana Jana Daciuka [10]. Wymusiło to stosowanie formatu danych proponowanego przez autora „kompresora".

Program budujący automat przyjmuje jako wejście plik z wyrazami. Każdy wyraz znajduje się w osobnym wierszu. Aby zapewnić możliwość łatwego porównania wyników pomiędzy programem Aspell a automatem skończonym, automat korzystał ze zbioru wyrazów słownika alternatywnego. Uzyskany plik automatu zawiera zakodowane 113 313 stanów i 318 129 przejść. Szczegółowy format przechowywania danych nie ma tutaj dużego znaczenia, ale warto pokrótce opisać, jak wygląda automat skończony dla słownika i w jaki sposób odbywa się poszukiwanie i korekta wyrazów.

Rysunek 3.1 przedstawia przykład automatu, który mógłby powstać dla wyrazów: *ala*, *alga*, *atom*, *butom*.

Algorytm dokonujący sprawdzenia, czy wyraz istnieje, przechodzi do stanu początkowego. Następnie stara się odnaleźć przejście pasujące do pierwszej litery. Załóżmy, że sprawdzamy wystąpienie w słowniku wyrazu *alfa*. Ponieważ istnieje łuk dotyczący *a*, algorytm przechodzi do wskazywanego przez niego stanu. W kolejnym kroku poszukuje przejścia związanego z literą *l*. Znajduje je i przechodzi do kolejnego stanu. Tym razem jednak nie znajduje żadnego przejścia dotyczącego litery *f*. Oznacza to, że szukany wyraz nie występuje w słowniku. Wyraz zostaje uznany za odnaleziony, tylko wtedy, gdy przy przejściu z jego ostatniego znaku automat trafi na łuk końcowy (na rysunku oznaczony grubszą strzałką).



Rysunek 3.1. Przykład automatu skończonego kodującego kilka wyrazów

Duże znaczenie dla szybkości działania ma kolejność testowanych przejść z danego stanu. Jak zauważono w pracy [7], zastosowanie sortowania przejść względem liczby wyrazów, do których dalej prowadzą pozwala przyspieszyć działanie algorytmu o 1/3 (w porównaniu z zastosowaniem kolejności alfabetycznej). Program generujący automat posiadał opcję zapewniającą takie sortowanie, więc została ona włączona.

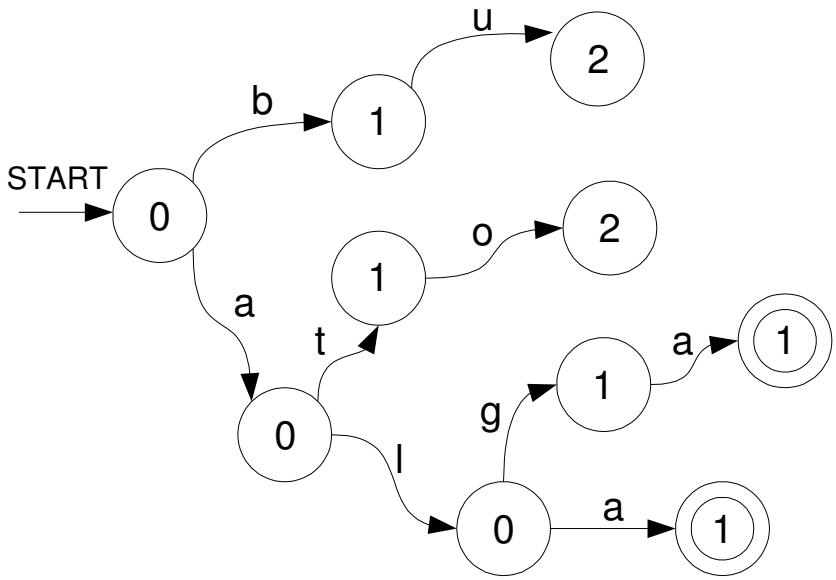
Testowanie występowania wyrazu w słowniku to nie koniec możliwości przedstawianego automatu skończonego. W pracach [25] i [8] przedstawiono algorytmy, dzięki któremu automat potrafi dokonywać bardzo wydajnej korekty ortograficznej. Są one wykorzystywane przez programy `fsa_spell` i `fsa_accent` z [10].

Prześledźmy w dużym skrócie sposób działania algorytmu wybierającego wyrazy ze słownika mogące stanowić poprawną wersję wyrazu nie występującego w słowniku. Posłużmy się wyrazem *alfa* z wcześniejszego przykładu i załóżmy, iż użytkownik dokonał w wyrazie co najwyżej jednego błędu podstawowego (odległość edycyjna wyrazu poprawnego i błędnego wynosi 1).

Błąd podstawowy to: dodanie nowego znaku, usunięcie znaku, zamiana dwóch znaków wyrazu miejscami lub zastąpienie znaku innym. Wykazano, iż większość błędów popełnianych w trakcie wpisywania tekstów na maszynie (lub przy użyciu komputera) to właśnie błędy podstawowe. Wynikają one ze sposobu wpisywania tekstu. Błąd podstawowy obejmuje swym zasięgiem również pojedyncze błędy ortograficzne w wyrazie.

Rysunek 3.2 przedstawia graf wyszukiwania poprawnych wersji dla wyrazu *alfa* przy założeniu stosowania automatu skończonego przedstawionego na rysunku 3.1. Wartości podawane wewnątrz stanów to wyliczona dla danej pozycji odległość edycyjna. Stany z

podwójną obwódką to stany, dla których algorytm zwróci propozycję poprawy wyrazu. Algorytm znajduje propozycje *ala* i *alga*.



Rysunek 3.2. Graf wyszukiwania propozycji poprawienia wyrazu alfa

Algorytm przeszukuje wszystkie możliwe odgałęzienia, o ile mogą one prowadzić do uzyskania wyrazu o odległości edycyjnej 1. Gdy tylko znaleziona propozycja ma wartość edycyjną większą od jedność, poszukiwania dalszych odgałęzień wychodzących w sprawdzanym kierunku nie są analizowane. Zwrócone zostają te wyrazy, które zakończyły się przy wartości odległości edycyjnej równej 0 lub 1.

Im większą wartość odległości edycyjnej uzna się za poprawną, tym więcej propozycji wyrazów zwróci algorytm. Będzie to jednak okupione znacznie większym czasem analizy wynikającym z potrzeby przetestowania bardziej rozbudowanego fragmentu grafu. Dla odległości edycyjnej 2 czas podania wszystkich możliwych rozwiązań potrafi wydłużyć się kilkukrotnie a dla odległości edycyjnej 3 nawet kilkunastokrotnie [25]. W implementowanym algorytmie na stałe zostało zaszyte korzystanie tylko i wyłącznie z odległości edycyjnej 1, by uzyskać jak najkrótszy czas działania.

Co istotne, uproszczenie i zmodyfikowanie algorytmu korekty ortograficznej pozwala dokonywać bardzo szybkiej korekty diakrytyki przy wykorzystaniu danych z tego samego słownika [10]. Znacznie zwiększona szybkość korekty diakrytycznej wynika z faktu, iż analizuje się w niej znacznie mniejszy fragment grafu — niejednokrotnie zawierający jedynie kilka dodatkowych odgałęzień w porównaniu z algorytmem sprawdzania wystąpie-

nia wyrazu w słowniku. Algorytm diakrytyki dokonuje dodatkowych testów dotyczących odgałęziania tylko dla znaków mających swe odpowiedniki z „ogonkami". Algorytm nie wykorzystuje odległości edycyjnej — zwraca wszystkie znalezione propozycje poprawy wyrazu.

3.3.4. Niejednoznaczność korekty ortograficznej

Należy zdawać sobie sprawę z możliwości wprowadzania znacznego szumu przez korektę ortograficzną. Opisane algorytmy korekcyjne potrafią w najgorszych przypadkach zwrócić nawet kilkadziesiąt propozycji poprawy błędnego wyrazu. Co więcej, poprawna wersja niekoniecznie musi wystąpić w propozycjach. Bez analizy kontekstu trudno ze zwróconych propozycji wybrać poprawną.

Sprawdźmy, jak system radziłby sobie z kilkoma różnymi błędami. Wyniki zostały uzyskane w programie `fsa_spell` z wykorzystaniem automatu skończonego przygotowanego na podstawie słownika alternatywnego. Dla wyrazu *testumemy* program zwraca tylko jedną propozycję — *testujemy* — która okazuje się być poprawną wersją. Dla wyrazu *maeka* program generuje 12 propozycji, między innymi: *macka*, *mleka*, *matka*, *mapka* i *marka*. Bez znajomości kontekstu trudno ustalić poprawną wersję w sposób automatyczny. W tekście dotyczącym wycieczek poprawny byłby wybór słowa *mapka* natomiast w tekście dotyczącym samochodów — *marka*.

Przed dokonaniem analizy ortograficznej warto spróbować uzupełnić znaki diakrytyczne dla danego wyrazu. Z dużą dozą prawdopodobieństwa uzyska się w takiej sytuacji wyraz, który miał na myśli autor tekstu. Program `fsa_spell` (pełna korekta ortograficzna) dla wyrazu *gorka* zwraca poza najprawdopodobniej poprawnym *górka* jeszcze 16 innych propozycji (*worka*, *korka* itp.). Program `fsa_accent` (uzupełnianie diakrytyki) zwraca tylko dwie propozycje: *górką* i *górka*.

Z powodu niemożliwości łatwego stwierdzenia, która wersja wyrazu jest najbardziej poprawna, system weryfikacji przekazuje do dalszej analizy wszystkie rozpoznane wersje. Określanie kontekstu można by spróbować wprowadzić na podstawie zebranej bazy wiedzy, gdyby istniał w systemie wyraźny podział na fazę uczenia i fazę testowania. Z powodu inkrementacyjnego charakteru weryfikacji komentarzy zrezygnowano z próby tworzenia takiego algorytmu. Niejednoznaczności powstałe po korekcie powinny do pewnego stopnia być znoszone przez algorytm klasyfikacji.

3.4. SPROWADZANIE WYRAZU DO POSTACI SŁOWNIKOWEJ (LEMATYZACJA)

3.4.1. Ogólna koncepcja

Większość systemów klasyfikacji tekstów wykorzystuje na etapie przygotowywania słów analizę morfologiczną, by w ten sposób zmniejszyć długość wektora cech i poprawić jakość klasyfikacji. Najczęściej nie jest to pełna analiza morfologiczna, ale sprowadzanie wyrazów do ich postaci słownikowych.

Język angielski posiada bardzo prostą fleksję, co pozwoliło na powstanie szybkiej, algorytmicznej wersji usuwania z wyrazów końcówek lub zastępowania fragmentów wyrazów (algorytm Portera, [27]). W przypadku języka polskiego, który posiada bardzo rozbudowaną fleksję, tego rodzaju algorytmiczne podejście jest bardzo trudne w realizacji. Najczęściej więc analizę morfologiczną realizuje się na podstawie indeksów a tergo, słowników lub innych rozwiązań zaprezentowanych w pracach [33], [35] i [36].

System automatycznej weryfikacji nie wymaga dokonywania złożonej analizy wyrazów. Najważniejsze jest zmniejszenie w możliwie najlepszy i najszybszy sposób fleksyjności słów. Bardzo proste rozwiązanie polega na utworzeniu w bazie danych tabeli zawierającej dwie kolumny: w pierwszej odmienioną formę słowa a w drugiej podstawową. Niestety trzeba tu pamiętać o ogromnej liczbie odmienionych słów dla języka polskiego (3,2 miliona w słowniku alternatywnym).

W ramach testu dokonano utworzenia w bazie danych MySQL 5 opisywanej tabeli na podstawie słownika alternatywnego. Powstała tabela wraz z indeksem zajmowała na dysku twardym komputera 180 MB (90 MB dane i 90 MB indeks). Czy można w jakiś sposób uzyskać podobny efekt i jednocześnie znacząco zmniejszyć wymagania dotyczące miejsca? Okazuje się, że tak.

3.4.2. Lematyzator wykorzystujący automat skończony

Deterministyczne automaty skończone w bardzo dobry sposób pozwalają skompresować słownik. System sprowadzania słów do formy podstawowej również zawiera pełną listę słów, która dodatkowo jest uzupełniona informacjami na temat formy podstawowej (w przypadku analizy morfologicznej także innymi danymi). Jan Daciuk w swoim zbiorze programów do tworzenia automatów skończonych ([10]) zastosował zmodyfikowany sposób zapisu automatu, by móc do każdego zakodowanego wyrazu dołączać dane morfolo-

giczne. Fakt ten wykorzystał Dawid Weiss, tworząc w języku Java lematyzator dla języka polskiego [36]. Lematyzator korzysta z automatu skończonego zawierającego pary form wyrazowych zebrane w słowniku Ispell-pl. Dzięki temu pełny słownik par form zajmujący 40 MB został „skompresowany" do 1,2 MB. Warto podkreślić, że zastosowanie tej postaci danych zapewnia ogromną szybkość działania (jest nieznacznie wolniejsze od sprawdzania istnienia wyrazu w słowniku). Wojciech Rutkowski na podstawie prac Jana Daciuka i Dawida Weissa dokonał na potrzeby projektu SENECA konwersji lematyzatora na język PHP 4 [29], postanawiając jednak nadal korzystać z pliku automatu skończonego uzyskanego ze słownika Ispell.

Z racji dostępności kodu lematyzatora w PHP 4 postanowiono skorzystać z tego rozwiązania, modyfikując je nieznacznie i tworząc przygotowany od nowa automat skończony dla lematyzatora na podstawie słownika alternatywnego.

3.4.3. Niejednoznaczności lematyzacji

Podobnie jak to miało miejsce w przypadku korekty ortograficznej, także na etapie lematyzacji pojawiają się niejednoznaczności. Jedno słowo może posiadać kilka form prostych w zależności od kontekstu. Przykładowo, słowo *mam* dotyczyć może formy podstawowej *mieć* („mam samochód") lub *mama* („tych mam"). Z powodu niemożności łatwego wybrania odpowiedniej wersji w sposób automatyczny, system zwraca wszystkie możliwe formy podstawowe dla danego wyrazu. Przedstawionych sytuacji nie jest dużo, więc nie powinny one wpłynąć znaczącą na jakość klasyfikacji.

3.5. WYKRYWANIE WULGARYZMÓW

Ostatnim aspektem dotyczącym fazy przygotowawczej jest wykrywanie wystąpienia w tekście wyrazów wulgarnych. Ponieważ zdecydowana większość witryn internetowych nie toleruje wulgaryzmów, wykrycie chociażby jednego wulgaryzmu powoduje odrzucenie komentarza bez jego dalszego przetwarzania.

Wydawać by się mogło, że wykrycie wulgaryzmu należy dokonywać na samym początku przetwarzania komentarza. Nie jest to prawdą. W takim podejściu użytkownik mógłby w łatwy sposób obejść zabezpieczenie, przekręcając wulgarny wyraz. Warto wcześniej dokonać poprawy ortograficznej, by w jak najlepszy sposób uniknąć przedostania się wulgaryzmu. Dodatkowo, by nie przechowywać wszystkich możliwych form wyrazu wul-

garnego, sprawdzenie wystąpienia wykonuje się jako ostatnią operację w całym ciągu przetwarzania.

W niniejszej pracy została wykorzystana lista wyrazów wulgarnych udostępniania w słowniku alternatywnym języka polskiego w źródłach danych kierowanych dla użytkowników programu Ispell [3]. Zawiera niecałe 400 wyrazów uznanych za wulgarne. Ponieważ oryginalna lista zawiera dodatkowo informacje o tworzeniu odmienionych form słów z formy podstawowej, zostały one usunięte (nie są potrzebne, gdyż lematyzator dokona odpowiedniej konwersji do formy podstawowej). Dodatkowo usunięto kilka bardzo łagodnych stwierdzeń.

3.6. IMPLEMENTACJA PRZEDSTAWIONYCH ZAGADNIEŃ I WYNIKI CZASOWE

Niniejszy podrozdział w bardziej szczegółowy sposób omawia implementację poszczególnych zagadnień opisanych we wcześniejszy podrozdziałach. Poszczególne punkty poświęcone zostały klasom wykonanym w języku PHP 5 i plikom pomocniczym przygotowanym na potrzeby projektu automatycznej weryfikacji komentarzy.

3.6.1. Pliki słowników FSA, stoplisty i wulgaryzmów

Opis powstania pliku stoplisty (*stoplist.txt*) znajduje się w podrozdziale 3.2. W pliku tym każdy wyraz zajmuje osobny wiersz.

Plik listy wyrazów wulgarnych (*vulgarism.txt*) używany do sprawdzania, czy w uzyskanym zbiorze słów znajdują się wulgaryzmy, został opisany w podrozdziale 3.5. Dodatkowo na podstawie tego pliku wykonano automat skończony, używając do tego celu programu `fsa_build` [10].

Dla programu Aspell 0.50 zainstalowano słownik alternatywny w odpowiedniej wersji bez dokonywania w nim żadnych modyfikacji.

Ponieważ dane dla lematyzacji nie były dostępne bezpośrednio, skorzystano z programu Aspell w wersji 0.60 oraz danych słownika alternatywnego dla systemu Myspell. Dane te składają się z dwóch plików: listy słów w formach podstawowych wraz z oznaczeniami odmiany oraz pliku informującego o sposobach odmiany słów języka polskiego (afiksach). Słownik alternatywny zawiera 221 tysięcy form podstawowych, które niekoniecznie są formami podstawowymi zgodnymi z oficjalnymi słownikami języka polskiego. Program Aspell w wersji 0.60 posiada opcję dokonującą rozwinięcia podanej formy podstawowej z oznaczeniami odmian na formy odmienione. Po wykonaniu prostego skryptu

wykorzystującego tę opcję powstał plik zawierający w poszczególnych wierszach formę podstawową i oddzielone spacjami formy odmienione. Kilka kolejnych skryptów (wykonanych przez autora pracy oraz udostępnionych przez Jana Daciuka) pozwoliło uzyskać plik słów odpowiedni do zastosowania w programie `fsa_build`. Powstały tak naprawdę dwa pliki „kompresowane do automatów" skończonych: pierwszy tylko i wyłącznie z listą słów (używany przez korektę ortograficzną i diakrytyczną) oraz drugi z dodatkową informacją o formie podstawowej (wykorzystywany przez lematyzator).

Program `fsa_build` na podstawie wspomnianych plików wygenerował dwa automaty skończone. Automat dla słownika ortograficznego zajmuje 1,2 MB. Zawiera 113 313 stanów i 318 129 przejść. Automat dla lematyzatora zajmuje 6,1 MB. Zawiera 935 112 stanów i 1 602 624 przejść. Ogromna różnica w liczbach wynika nie tyle z faktu istnienia dużej ilości dodatkowych informacji związanych z lematyzacją, ale z niemożliwości łatwego współdzielenia stanów i przejść w drugim automacie z powodu różnic w formach podstawowych.

Kolejny skrypt przygotowany przez autora pracy posłużył do umieszczenia danych lematyzacji w tabeli bazy danych MySQL 5. Powstała tabela zajmowała 180 MB, przy czym połowę tej wartości stanowił indeks klucza głównego założonego na kolumnie z odmienionymi formami słów.

3.6.2. Klasy słowników FSA

Na potrzeby niniejszej pracy powstało łącznie 7 plików PHP z klasami wykorzystującymi automaty skończone do wykonywania operacji na wyrazach. Klasy te były w dużej mierze wzorowane na algorytmach i kodzie w języku C++ autorstwa Jana Daciuka [10]. Niezoptymalizowana wersja klasy lematyzatora FSA to poza niewielkimi zmianami kod autorstwa Wojciecha Rutkowskiego udostępniony za jego zgodą do wykorzystania w tej pracy.

Klasa `Fsal` (lematyzator FSA) zawiera następujące modyfikacje w porównaniu z oryginalną wersją:

- jest w całości przeniesiona do języka PHP w wersji 5 (oryginał został napisany w PHP 4);
- korzysta z innej zakodowanej na stałe wartości GLT (liczby bajtów używanych do określenia pozycji stanu wskazywanego przez analizowany łuk);
- dodatkowe komentarze;

– usunięte metody dotyczące obrazowania przejścia przez automat.

Klasa `Fsaa` (korektor diakrytyki FSA) została napisana przez autora pracy na podstawie kodu w języku C++. Ponieważ język PHP nie zawiera preprocesora i jest znacząco wolniejszy, dokonano pewnych uproszczeń (zmniejszenia elastyczności rozwiązania) i modyfikacji algorytmu, by zapewnić jego dużą szybkość. Powstały algorytm w odróżnieniu od oryginału potrafi korzystać tylko z jednego formatu danych automatu skończonego.

Klasa `Fsas` (korektor ortograficzny FSA) również został napisany na podstawie kodu w języku C++ z zastosowaniem wielu uproszczeń i modyfikacji. Jednym z podstawowych uproszczeń jest fakt, iż potrafi dokonywać korekty tylko i wyłącznie dla odległości edycyjnej 1. Pozostawiono jednak pewien ciekawy element oryginalnego algorytmu Daciuka a mianowicie traktowanie w korekcie ortograficznej litery *rz* jako *ż* a także liter *om*, *oł*, *on* jako *ą*. Zwiększa to złożoność algorytmu (wymaga sprawdzania dodatkowych odgałęzień), ale pozwala lepiej radzić sobie z błędami orograficznymi.

Ponieważ oryginalne rozwiązanie w języku C++ wykorzystywało preprocesor i rozwijanie funkcji w miejscu wywołania w celu zwiększenia szybkości działania, postanowiono skorzystać z podobnego rozwiązania w języku PHP. Język ten nie obsługuje automatycznego rozwijania funkcji, gdyż jest językiem skryptowym. Co więcej, każda próba wywołania funkcji w większości sytuacji wymaga wyszukania adresu funkcji na podstawie jej nazwy [2]. Zapewnia to ogromną elastyczność (można w locie tworzyć nazwy funkcji), ale drastycznie ogranicza wydajność. Postanowiono wykonać trzy nowe wersje klas `Fsas`, `Fsal` i `Fsaa` (o tych samym nazwach, ale w innych plikach) przy zastosowaniu ręcznego rozwinięcia najbardziej newralgicznych metod. Podejście to pozwoliło na niemal dwukrotny wzrost szybkości działania algorytmów FSA. Niestety ręczne rozwinięcia sprawiły, iż algorytm stał się znacznie mniej czytelny.

Na podstawie zoptymalizowanej wersji kodu klasy `Fsas` wykonano dodatkową klasę (`Fsav`) zapewniającą proste sprawdzanie, czy przekazany wyraz jest wyrazem wulgarnym. Klasa została całkowicie pozbawiona elementów związanych z korektą i została zmodyfikowana na potrzeby mniejszej wartości GLT (długości adresu skoku zawarta w łuku).

3.6.3. Klasy tokenizacji i walidacji oraz klasy pomocnicze

Wykonana klasa walidacji danych (`Validation`) poszukuje w tekście adresów email i URL w sposób opisany we wcześniejszym podrozdziale. Zostały w tym celu napi-

sane dwie metody statyczne korzystające z funkcji dotyczących wyrażeń regularnych w stylu Perla. Dodatkowo klasa wczytuje plik z listą wulgaryzmów i zapewnia sprawdzenie, czy w podanej tablicy wyrazów znajduje się niedozwolone słowo.

Utworzona klasa tokenizera (`Tokenizer`) łączy w sobie wiele zadań. Dokonuje przetworzenia oryginalnego tekstu na listę słów, stosując algorytm o liniowej złożoności i usuwając wszystkie znaki nie będące literami i cyframi. Dodatkowo wykorzystuje opisaną wcześniej stoplistę. Oto pełna lista jej zadań:

- zamiana wielkich liter na małe,
- usuwanie powtórzeń liter, by na przykład z tekstu *kkkkaaallkaa* uzyskać *kalka*,
- pomijanie wyrazów jednoznakowych i dwuznakowych (w języku polskim są to najczęściej skróty, spójniki, zaimki itp.),
- niezależnie traktowanie liczb (liczby są wydobywane z tekstu a nie pomijane, jak to ma miejsce w wielu innych systemach kategoryzacji),
- zamiana liczb na wyrazy określające przedział wartości (zastosowano bardzo prosty algorytm opisany we wcześniejszym podrozdziale),
- jeśli przy inicjalizacji został podany plik stoplisty, usuwa wyrazy ze stoplisty,
- zwraca informację o potencjalnych wulgaryzmach (liczba wystąpień co najmniej czterech **** pod rząd).

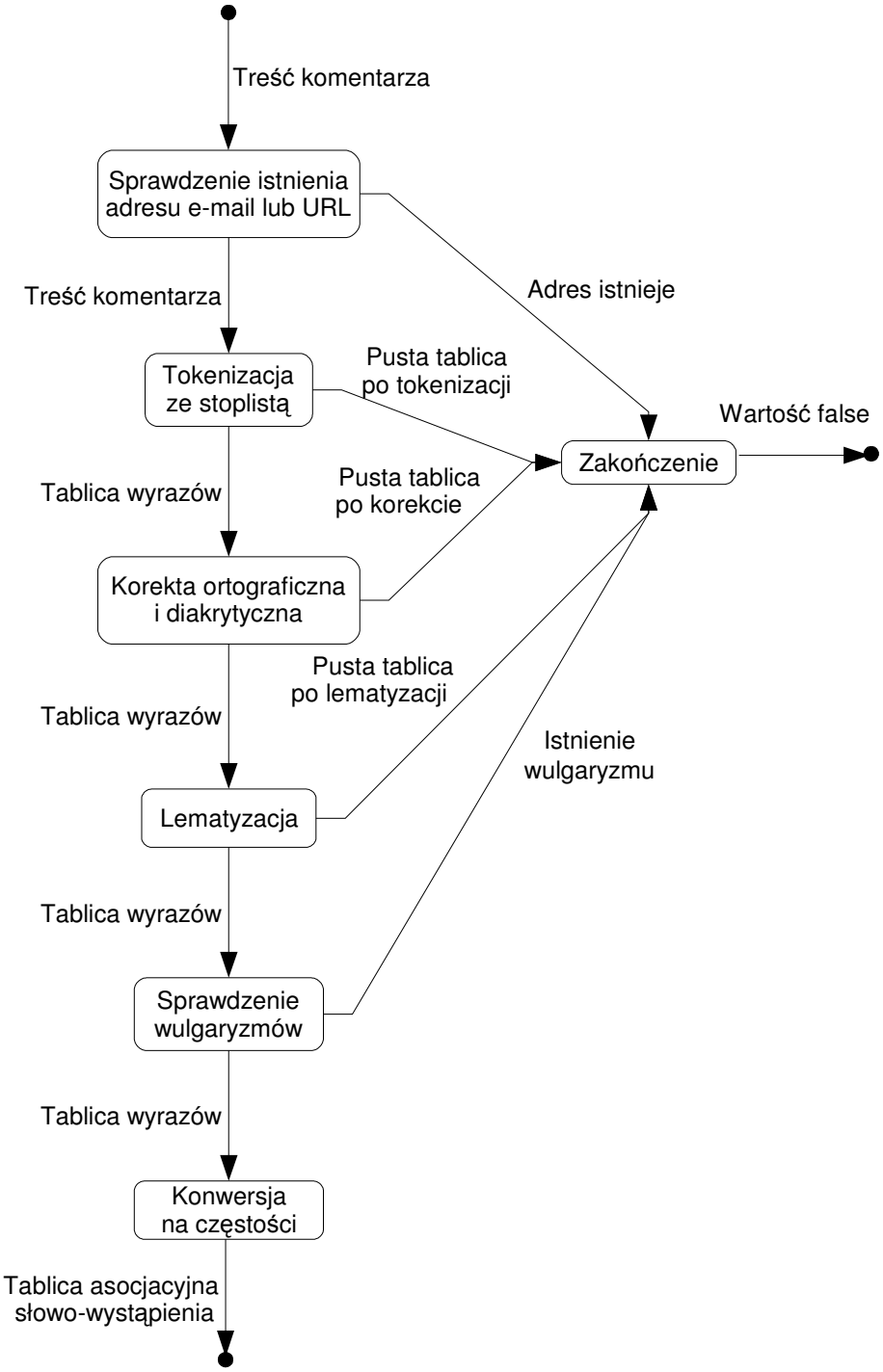
Klasa pomocnicza `Timer` wykonana na potrzeby pracy zapewnia dokładny pomiar czasu, który upłynął między dwoma zadanymi punktami. Klasa pozwala mierzyć łączny czas wykonania danego fragmentu kodu nawet wtedy, gdy znajduje się w pętli (sumuje czasy pojedynczych wykonań). Dodatkowa statyczna metoda konwertuje czas na minuty i sekundy.

Klasa pomocnicza `DBHelper` zapewnia metody statyczne upraszczające inicjalizację i wykonywanie kilku prostych operacji pobierania danych z bazy danych MySQL.

3.6.4. Abstrakcyjna klasa klasyfikatora

Ponieważ wszystkie wykonywane klasyfikatory korzystają z wektora cech (listy słów wraz z częstością ich wystąpienia) i stosują podobny interfejs, postanowiono zapewnić dla nich jedną klasę bazową. Klasa `Classify` definiuje szczegółowy kod zajmujący się obróbką wstępną tekstu (tokenizacja, ortografia, lematyzacja) i metody abstrakcyjne przygotowane do uzupełnienia przez konkretne klasyfikatory. Klasa inicjalizuje obiekty automatów skończonych, połączenie z bazą danych, deskryptor korzystania z systemu Aspell itp.

Rysunek 3.3 przedstawia kolejne etapy obróbki tekstu przez metodę `doPreparation()` wraz ze zwracanymi typami wyników.



Rysunek 3.3. Etapy przygotowania komentarza

Korekta ortograficzna składa się z kilku etapów. Najpierw zostaje sprawdzone, czy wyraz istnieje w słowniku. Jeśli tak, jest dołączany do listy słów. W przeciwnym razie

dochodzi do korekty diakrytycznej. Jeśli zwróci co najmniej jeden wynik (a może i kilka), jest on wpisywany do listy słów. W przeciwnym razie dochodzi do pełnej próby korekty ortograficznej. Kod pobiera pięć pierwszych proponowanych popraw (zwróconych może być nawet kilkadziesiąt) i wpisuje je do listy słów. Gdy żadna propozycja nie zostanie zgłoszona, w zależności od ustawień pomija wyraz lub też przenosi go w oryginalnej postaci do dalszych etapów analizy.

3.6.5. Testy wydajności i wybór ostatecznego rozwiązania

Wszystkie przedstawione wyniki testów zostały przeprowadzone w systemie Windows XP na procesorze Athlon 64 3700+ (maksymalne taktowanie 2,4 GHz) z użyciem PHP w wersji 5.1.4 (włączony Zend Optimizer).

Pierwszy test wyboru wydajniejszego rozwiązania dotyczył porównania szybkości sprawdzania wystąpienia wyrazu wulgarnego. Jedno podejście stosuje napisaną w PHP 5 klasę `Fsav` z automatem skończonym przechowującym 361 wyrazów wulgarnych. Drugie podejście korzysta z wczytanej do pamięci tablicy wyrazów i funkcji PHP `in_array()` sprawdzającej wystąpienie wskazanej wartości w zadanej tablicy. Test sprawdzający kilkanaście wyrazów (w tym dwa wulgarne) wykazał, że rozwiązanie z pełnym wyszukiwaniem wartości w tablicy (0,14 ms) jest około dwukrotnie szybsze od automatu skończonego (0,29 ms). Wynik może wydawać się nieprawdopodobny. Automat skończony dokonuje znacznie mniej testów niż przeszukanie tablicy nie zakładające posortowania zawartych w niej danych. Warto jednak wspomnieć, iż funkcja PHP `in_array()` jest zaimplementowana w języku C a cały automat w języku PHP. To właśnie powolność interpretowanego PHP i mały rozmiar zbioru wyrazów zadecydowała o takim a nie innym wyniku. Im wyrazów do sprawdzenia byłoby więcej, tym większy zysk z korzystania z automatu skończonego na etapie wyszukiwania. Ostateczne rozwiązanie stosuje zwykłe sprawdzanie wystąpienia wyrazu w tablicy.

Kolejny test rozstrzygający wybór podejścia, który miał się odbyć, w zasadzie stał się zbędny. Miał on polegać na porównaniu szybkości lematyzacji FSA z naiwną lematyzacją wykorzystującą pełny zbiór danych zawarty w tabeli bazy danych. Okazało się, że różnica w rozmiarach obu rozwiązań jest ogromna (180 MB baza danych, 6,1 MB automat skończony lematyzacji), by ktokolwiek stosował w typowej witrynie podejście bazodanowe. Z ciekawości dokonano kilkukrotnego sprawdzenia szybkości zwrócenia wyników dla kilkunastu wyrazów. Co nie dziwi, rozwiązanie FSA okazało się znacznie wydajniejsze

(czas sprawdzenia kilkunastu wyrazów poniżej 1 ms) w porównaniu z rozwiązaniem bazodanowym (nawet do 300 ms). Ostateczne rozwiązanie stosuje lematyzację FSA.

Następny test dotyczył porównania wersji standardowych klas FSA z wersjami zoptymalizowanymi przez ręczne rozwinięcie niektórych metod w miejscu ich wywołania. Tabela 3.1 przedstawia zsumowane wyniki uzyskane dla testowania 10 kilkunastowyrazowych komentarzy podzielone według faz korekty — ortografia i lematyzacja. Faza ortografii wykorzystuje dwie klasy FSA (`Fsas` i `Fsaa`).

Tabela 3.1. Szybkość działania zwykłych i zoptymalizowanych klas FSA

Wersja klasy / **Faza korekty**	**Podstawowa**	**Zoptymalizowana**	**Przyspieszenie**
Korekta ortograficzna	3,718 s	1,784 s	2,11x
Lematyzacja	0,309 s	0,121 s	2,55x

Z przedstawionych danych jednoznacznie wynika, że ręczne rozwinięcie przynosi znaczący, bo ponad dwukrotny, wzrost szybkości działania klas FSA. Z tego powodu kolejny test stosuje zoptymalizowane wersje klas.

Ostatni test ma za zadanie porównać szybkość pełnej korekty ortograficznej (bez uzupełniania diakrytyki, bo nie jest ona obsługiwana przez Aspell) przeprowadzanej z użyciem modułu `pspell` (umożliwia dostęp do biblioteki Aspell) i zoptymalizowanej klasy `Fsas`. Tabela 3.2 przedstawia wyniki uzyskane dla 20 pierwszych tekstów pozytywnych wszystkich trzech zestawów komentarzy.

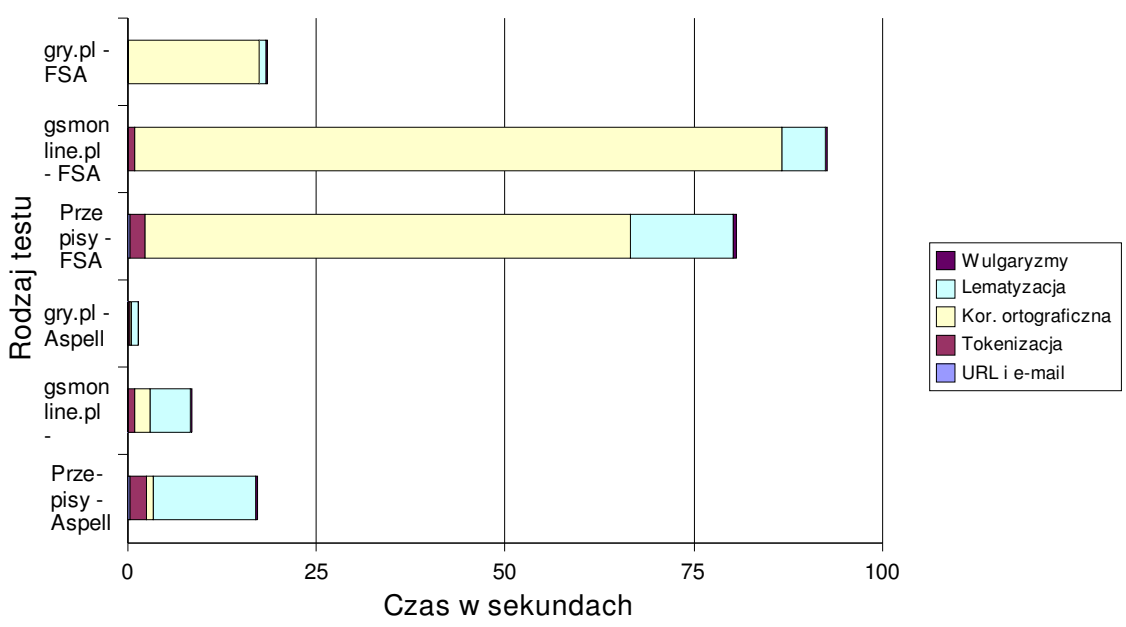
Tabela 3.2. Porównanie szybkości działania Aspell i FSA

Zestaw koment. / **Korektor**	**gry.pl**	**gsmonline.pl**	**Kucharskie**
Aspell w trybie `FAST`	0,009 s	0,101 s	0,024 s
Zoptymalizowany FSA	1,012 s	13,944 s	1,979 s
Różnica	*1120%*	*1380%*	*820%*

Biblioteka Aspell w tym porównaniu jest średnio 10 razy szybsza od zoptymalizowanej wersji FSA. Przedstawiane czasy łączą w sobie sprawdzanie poprawności wyrazu i zbierania sugestii przy czym test poprawności wyrazu przeprowadzany jest dla każdego słowa a sugestie tylko dla słów niepoprawnych. Tutaj ponownie o wyniku w ogromnym

stopniu zadecydowała implementacja FSA w języku skryptowym PHP, który jest znacznie wolniejszy od Aspell zaimplementowanego w C. Warto odnieść te wyniki do wyników przedstawionych w pracy [8]. Tam algorytm FSA (choć w nieco innej postaci) zaimplementowany w C okazał się dwukrotnie szybszy od biblioteki Aspell w trybie `FAST` w przypadku zwracania pierwszych 10 sugestii (1,93 ms i 4,2 ms). Wysnuć stąd można wniosek, iż implementacja FSA w PHP jest co najmniej 100 wolniejsza od implementacji w C. Ostateczne rozwiązanie stosuje bibliotekę Aspell. Klasy `Fsaa` użyto do przeprowadzania korekty diakrytycznej z racji niedostępności tego rodzaju rozwiązania w bibliotece Aspell.

Rysunek 3.4 przedstawia rozkład czasu poświęcanego na poszczególne etapy przetwarzania w końcowym rozwiązaniu stosującym optymalne algorytmy opisane we wcześniejszych akapitach. Do testów wykorzystano 500 tekstów pozytywnych z każdej grupy komentarzy.



Rysunek 3.4. Rozkład czasów poszczególnych etapów przygotowywania tekstu dla różnych zbiorów komentarzy i algorytmów korekty ortograficznej

Warto zwrócić uwagę, iż znaczącymi elementami na tym etapie są czasy korekty ortograficznej, która w przypadku korekty FSA znacząco dominuje, i lematyzacji, która staje się najbardziej czasochłonnym elementem w momencie stosowania Aspell. Etap tokenizacji zyskuje na znaczeniu w przypadku dłuższych testów. Minimalne znaczenie mają testy

wulgaryzmów oraz adresów URL i e-mail (choć są w tym celu używane wyrażenia regularne).

4. METODY KLASYFIKACJI TEKSTÓW

4.1. KLASYFIKACJA TEKSTU

4.1.1. Definicja

Klasyfikacja tekstów to sposób automatycznego przypisywania tekstu na podstawie jego zawartości do predefiniowanych kategorii (nazywanych też klasami). W większości systemów klasyfikacji stosuje się wiele kategorii, które mogą (choć nie muszą) się na siebie częściowo nakładać. Weryfikacja komentarzy korzysta tylko z dwóch rozłącznych kategorii — komentarz jest poprawny i powinien zostać wyświetlony lub też jest nie poprawny i należy go odrzucić.

Klasyfikatorem nazywa się funkcję, która dla zadanego dokument przedstawianego najczęściej jako wektor cech przypisuje po jednej wartości z przedziału [0, 1] na każdą kategorię. W podstawowej klasyfikacji binarnej dokument zostaje przypisany do tej kategorii, która uzyska z klasyfikatora większą wartość.

4.1.2. Uczenie maszynowe

Klasyfikator można tworzyć na dwa sposoby: manualnie lub automatycznie (dzięki uczeniu maszynowemu). W pierwszym z nich korzysta się z ekspertów z klasyfikowanej dziedziny i inżynierów potrafiących wykonać odpowiedni zestaw reguł zapewniający poprawną klasyfikację niespotkanych dotąd dokumentów. To podejście wymaga ogromnego nakładu pracy. Jest szczególnie nieefektywne, gdy trzeba często modyfikować zbiór kategorii. Co więcej, gdy kilka osób pracuje nad jednym projektem, potrafią pojawić się nieścisłości pomiędzy ekspertami.

W automatycznym podejściu do klasyfikacji tekstów wykorzystuje się algorytmy uczenia maszynowego, przedstawiając im przykłady dokumentów umieszczonych w poszczególnych kategoriach przez eksperta dziedzinowego. Jest to uczenie nadzorowane. Wymaga co prawda pewnej liczby sklasyfikowanych wcześniej dokumentów, ale nakład pracy związany z ich uzyskaniem jest znacząco niższy niż w przypadku ręcznego kreowania klasyfikatora. Co więcej, stosując uczenie inkrementacyjne zapewnia się dryft sposobu przypisywania dokumentów do kategorii. Raz napisany kod klasyfikatora można wykorzystać do klasyfikacji różnego rodzaju tekstów a nawet do identyfikacji języka, w którym tekst został napisany.

Niestety uczenie maszynowe ma pewne wady wynikające z konstrukcji samego języka naturalnego. Przykładowo, pewne kategorie można łatwo rozpoznawać już po zastosowaniu kilku wyrazów, inne mogą wymagać setek a nawet tysięcy wyrazów, by dobrze uchwycić znaczenie całej kategorii. Ekspert dziedzinowy łatwo mógłby dokonać odpowiedniego zwiększenia długości i szczegółowości opisu dla wymagających tego kategorii lub też wskazać wyrazy mogące powodować częste pomyłki. Uczenie maszynowe musi samo na podstawie częstości wystąpień wyrazów w zbiorze uczącym dokonać oceny, który wyraz jest istotny, a który nie.

Przedstawiony problem staje się szczególnie ważny, gdy jak w niniejszej pracy chce się zapewnić dobrą pracę systemu przy jak najmniejszym nakładzie pracy wstępnej i jednocześnie umożliwić ciągłe „doszkalanie" klasyfikatora.

4.1.3. Niejednoznaczności

Uzyskanie doskonałego jakościowo klasyfikatora wymaga utworzenia systemu rozumiejącego w pewien sposób język naturalny. Niestety poznanie dokładnego znaczenia zdań jest zadaniem szczególnie trudnym dla komputerów z racji wielu niejednoznaczności.

Jedno zdanie może mieć wiele znaczeń, szczególnie jeśli rozpatruje się nie ciąg ale zbiór słów (słowa połączone razem w odpowiedniej kolejności mają inne znaczenie niż gdyby podano je osobno). Rozważmy zdanie „Przeczytałem książkę od deski do deski". Ma ono znaczenie dosłowne i wynikające ze związku frazeologicznego. Nie zawsze suma znaczeń słów równa się znaczeniu całego zdania. Czasem odpowiednia kombinacja całkowicie je zmienia.

Synonimy również mogą w znaczący sposób utrudnić odgadnięcie rzeczywistej natury zdania, jeśli we wcześniejszych nauczonych dokumentach pojawiała się tylko jedna forma (na przykład *kolor*) a w analizowanym dokumencie pojawia się druga (*barwa*). W wielu językach zaleca się stosowanie synonimów, by nie powtarzać w kolejnych zdaniach tych samych wyrazów. W ten sposób klasyfikator ma utrudnione zadanie przy rozpoznawaniu kategorii dokumentu.

Kolejnym aspektem języka utrudniającym klasyfikację są słowa pisane dokładnie w ten sam sposób, ale posiadające kilka znaczeń. Przykładem może być słowo *piec*, które w zależności od kontekstu może dotyczyć innych kwestii: *piec żeliwny* lub *piec ciasto*.

Próby uwzględnienia rozumienia języka naturalnego przez klasyfikator zawsze zwiększają jego złożoność a co za tym idzie czas działania. W niniejszej pracy ogromne

znaczenie ma szybkość, więc tego rodzaju kwestie nie są rozważane ani implementowane bezpośrednio.

4.2. MIARY JAKOŚCI KLASYFIKATORÓW

Przed przedstawieniem klasyfikatorów, ich zalet i wad oraz osiąganych rezultatów, warto omówić powszechnie stosowane sposoby oceny jakości klasyfikatorów [30]. Jak się wkrótce okaże, nie są one w większości sytuacji odpowiednie do określania jakości prostej klasyfikacji binarnej zawierającej możliwość odmówienia przypisania dokumentu do konkretnej kategorii.

4.2.1. Pełność, precyzja, dokładność i błąd

Pełność (ang. *recall*) i precyzja (ang. *precision*) to terminy wywodzące się z klasycznego wydobywania informacji (IR) zaadoptowane na potrzeby klasyfikacji tekstów [1]. Precyzja oznacza prawdopodobieństwo, iż losowy dokument dostanie przypisany do zadanej kategorii i okaże się ona prawidłowa. Pełność oznacza prawdopodobieństwo, że dla losowego dokumentu, który powinien zostać umieszczony w zadanej kategorii, podjęto taką decyzję. Prawdopodobieństwa te wylicza się, korzystając z wzorów (4.1) i (4.2) oraz tabeli możliwości przedstawionej w tabeli 4.1.

$$recall = \frac{TP}{TP + FP} \tag{4.1}$$

$$precision = \frac{TP}{TP + FN} \tag{4.2}$$

Tabela 4.1. Tabela możliwości dla wyliczania pełności i precyzji

Dla konkretnej kategorii		**Ocena eksperta**	
		TAK jest poprawne	NIE jest poprawne
Ocena klasyfikatora	Przypisano TAK	*TP*	*FP*
	Przypisano NIE	*FN*	*TN*

Wyliczenia pełności i precyzji przeprowadza się najczęściej w systemach dokonujących klasyfikacji do wielu kategorii. W systemach dokonujących tylko klasyfikacji binarnej niejednokrotnie stosuje się inne wyliczenia a mianowicie dokładność (ang. *accuracy*) i błąd (ang. *error*). Wzory (4.3) i (4.4) dotyczą wyliczania tych dwóch wartości na podstawie tabeli możliwości.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.3}$$

$$error = \frac{FP + FN}{TP + FP + TN + FN} \tag{4.4}$$

Aby określać średnią dobroć klasyfikatora dla wszystkich kategorii, używa się dwóch metod uśredniania: mikro i makro. W metodzie makro najpierw wylicza się precyzję i pełność dla każdej kategorii a następnie uśrednia się ją, by uzyskać wartość globalną. W ten sposób każda kategoria ma to samo znaczenie niezależnie od tego, czy jest duża (zawiera więcej dokumentów), czy jest mniejsza. Metoda mikro polega na zsumowaniu wartości *TP*, *FP*, *TN* i *FN* dla wszystkich kategorii i następnie na podstawie tych danych wyliczenia precyzji i pełności. To podejście zapewnia tę samą wagę każdemu klasyfikowanemu dokumentowi — liczniejsze kategorie mają większy wpływ na końcową wartość.

4.2.2. Punkt przełamania i miara F

Wartości pełności i precyzji muszą być rozpatrywane wspólnie, ponieważ w oderwaniu od siebie nie pozwalają poprawnie określić dobroci klasyfikatora. Można łatwo wykonać klasyfikator zatwierdzający wszystkie dokumenty do kategorii i osiągający w ten sposób precyzję wynoszącą 100%. Dla takiego systemu pełność będzie jednak wyjątkowo niska. Z tych powodów w ocenie klasyfikatorów tekstu wykorzystuje się dwie miary wiążące ze sobą precyzję i pełność.

Najczęściej chce się uzyskać możliwie najlepszą precyzję i pełność, ale zwiększanie jednej z nich pociąga za sobą zmniejszanie drugiej. Punkt przełamania określa miejsce, w którym wartości te są sobie równe lub też znajdują się najbliżej siebie.

Wzór (4.5) opisuje miarę *F* stosowaną do łączenia precyzji i pełności. We wzorze tym parametr β pozwala przypisać różną wagę precyzji i pełności. Najczęściej β jest równe 1, czyli oba parametry w takim samym stopniu uczestniczą w uzyskiwanej wartości końcowej.

$$F_\beta = \frac{(\beta^2 + 1) * precision * recall}{\beta^2 * precision + recall} \tag{4.5}$$

Niestety różne prace stosują różne miary oceny przez co niejednokrotnie pojawiają się problemy z porównywaniem poszczególnych algorytmów.

4.3. KLASYFIKATORY REGUŁOWE

Klasyfikatory regułowe podzielić można na dwie podstawowe grupy. Pierwsza z nich to klasyfikatory z ręcznie tworzonymi regułami. Druga to klasyfikatory z automatyką uczącą korzystające z drzew decyzyjnych lub teorii dowodzenia. Obecnie tym algorytmom nie poświęca się zbyt wiele uwagi.

4.3.1. Klasyfikatory tworzone ręcznie

Tego rodzaju klasyfikatory powstają w sposób manualny przez określanie reguł podejmowania decyzji. Osobą dokonującą kreowania reguł jest najczęściej ekspert dziedzinowy. Z racji dużych kosztów tworzenia i obsługi tego rodzaju systemów, stosuje się je obecnie tylko do bardzo złożonych działań.

Przykładem najprostszego klasyfikatora regułowego tworzonego ręcznie może być system weryfikacji odrzucający komentarze zawierające wulgaryzmy. Listę wyrazów wulgarnych można traktować jako reguły „jeśli wystąpi, odrzuć; w przeciwnym razie pozostaw".

4.3.2. Drzewa decyzyjne

Istnieje kilka algorytmów budujących drzewa decyzyjne na podstawie zbioru uczącego: CART, C4.5, Ripper i CHAID [1]. Różnią się one przede wszystkim sposobem kreowania drzewa decyzyjnego.

Algorytm CART tworzy decyzyjne drzewo binarne, dzieląc zbiór wektorów testowych w każdym węźle na podstawie wybranego, pojedynczego elementu wektora. Wyboru elementu dokonuje się w taki sposób, by zapewnić podział dokumentów treningowych w jak największym stopniu. Podziały wykonuje się tak długo, aż nie będzie można znaleźć żadnego podziału zapewniającego odpowiednio mocne rozdzielenie dokumentów w analizowanym węźle. Dzięki temu procesowi każdy z dokumentów uczących zostaje przypisany do pewnego liścia drzewa decyzyjnego. Pozostaje przypisać każdemu liściu drzewa odpowiednią kategorię. Nie oznacza to, że wszystkie dokumenty uczące osiągające konkretny liść będą należeć do tej samej kategorii. W ten sposób powstają błędy.

Kolejny etap obróbki drzewa polega na jego „odchudzeniu", ponieważ w obecnej postaci drzewo decyzyjne bardzo dobrze radziłoby sobie z dokumentami treningowymi, ale wykazywałoby się słabą jakością klasyfikacji nieznanych do tej pory dokumentów. Mówi się w tym przypadku o „przesyceniu" drzewa zbiorem treningowym. Celem odchu-

dzenia drzewa jest usuniecie odgałęzień, które zapewniają najsłabszą siłę przewidująca u liścia. Podstawowym zadaniem tej operacji jest wykonanie drzewa, które będzie potrafiło dobrze klasyfikować nowe dokumenty.

Z przedstawionego opisu jednoznacznie wynika, że drzewa decyzyjne stosunkowo słabo nadają się do zastosowania w tworzonym systemie automatycznej weryfikacji komentarzy, gdyż nie ułatwiają inkrementacyjnego uzupełniania bazy wiedzy. Co więcej, wykonanie ich w taki sposób, by potrafiły wydajnie korzystać z przechowywania w informacji w bazie danych niekoniecznie jest proste.

4.4. KLASYFIKATORY LINIOWE

Obecnie do tej grupy zaliczyć można najwięcej klasyfikatorów. Grupa obejmuje zarówno klasyfikatory bazujące na teorii probabilistycznej (Naive Bayes, choć w pewnych pracach dla tego klasyfikatora tworzy się osobną grupę tak zwanych klasyfikatorów parametrycznych [30]), sieciach neuronowych i ich modyfikacjach (Winnow, Windrow-Hoff), na tworzeniu globalnego profilu dla każdej kategorii (Rocchio, Centroid, KAN i DCM) i na generowaniu wektora separującego (SVM). Niniejszy podrozdział pokrótce opisuje każde z rozwiązań i przyświecające mu cele bez porównywania poszczególnych rozwiązań (zostanie ono przeprowadzone w ostatnim podrozdziale).

4.4.1. Naive Bayes

Naiwny klasyfikator Bayesa [1], [37] to metoda uczenia maszynowego wykorzystywana do rozwiązywania problemu sortowania. Jest używana nie tylko w klasyfikacji tekstów. Głównym zadaniem klasyfikatora Bayesa jest przyporządkowanie nowego elementu do jednej z wcześniej zdefiniowanych klas decyzyjnych (kategorii).

Klasyfikator wykorzystuje zbiór danych treningowych do estymacji prawdopodobieństwa należenia nowego dokumentu do poszczególnych kategorii po podaniu zbioru jego cech. Klasyfikator nazywa się naiwnym, ponieważ zakłada niezależność poszczególnych słów w tekście, co oczywiście w większości sytuacji nie jest prawdą. Co ciekawe, nie obniża to znacząco jakości klasyfikacji. Do obliczenia estymacji prawdopodobieństwa wykorzystuje się podstawowy wzór teorii Bayesa (4.6).

$$P(c_j \mid \mathbf{d}) = \frac{P(c_j)P(\mathbf{d} \mid c_j)}{P(\mathbf{d})} \tag{4.6}$$

We wzorze **d** to analizowany dokument natomiast c_j to kategoria. Dzielnik w przedstawionym wzorze nie różni się w poszczególnych kategoriach, więc można go pominąć. Dodatkowo estymację ułatwia wspomniane wcześniej założenie o niezależności słów. Po uproszczeniach uzyskuje się wzór (4.7).

$$P(c_j \mid \mathbf{d}) = P(c_j)\prod_{i=1}^{M} P(d_i \mid c_j) \tag{4.7}$$

We wzorze M to liczba analizowanych słów dokumentu. Przybliżenie dla $P(c_j)$ uzyskuje się, dzieląc liczbę dokumentów treningowych przypisanych do danej kategorii przez łączną liczbę dokumentów treningowych. Przybliżenie dla $P(d_i|c_j)$ wylicza się, korzystając z wzoru (4.8).

$$\hat{P}(d_i \mid c_j) = \frac{1 + N_{ij}}{M + \sum_{k=1}^{M} N_{kj}} \tag{4.8}$$

W przedstawionym wzorze N_{ij} to liczba wystąpień, jaką słowo i wystąpiło w dokumencie kategorii c_j w zbiorze treningowym.

Na podstawie przedstawionego opisu i wzorów można wysnuć wniosek, iż klasyfikator ten nadaje się zarówno do uczenia inkrementacyjnego, jak i do łatwej implementacji przechowywania danych w bazie danych.

4.4.2. Rocchio

Podstawowy pomysł algorytmu Rocchio [4] polega na konstrukcji dla każdej kategorii wektora wag słów. Wektor ten budowany jest przez analizę słów występujących w poszczególnych dokumentach treningowych. Powstały dla kategorii wektor stanowi uogólnioną wersję wszystkich dokumentów należących do danej kategorii.

Algorytm w podstawowej wersji wykorzystuje miarę TFIDF do określania wag poszczególnych słów w analizowanym dokumencie. Miara ta przedstawiona wzorem (4.9) uwzględnia zarówno liczbę wystąpień słowa w analizowanym dokumencie, jak i częstość wystąpienia słowa we wszystkich dokumentach wykorzystywanych w trakcie uczenia klasyfikatora.

$$W_i = \mathrm{TF}(w_i, d) \log\left(\frac{D}{\mathrm{DF}(w_i)}\right) \tag{4.9}$$

W przedstawionym wzorze W_i to waga słowa dokumentu używana przez docelowy algorytm, TF to liczbą wystąpień słowa w_i w dokumencie d, D to liczba wszystkich dokumentów natomiast DF to liczba dokumentów, w których występuje słowo w_i.

Globalny wektor kategorii wylicza się, złączając (sumując) wszystkie wektory dokumentów należących do jednej kategorii. Następnie mnoży się wagę każdego dokumentu przez przyjęty współczynnik wpływu elementów pozytywnych α i dzieli się przez liczbę dokumentów należących do kategorii. Kolejny krok to odjęcie średniej wartości poszczególnych wag w wektorze dokumentów nie należących do analizowanej kategorii. Gdy wartość wagi któregoś ze słów stanie się ujemna, jest ustawiana na 0. Wzór (4.10) łączy wszystkie wspomniane elementy.

$$\vec{c}_k = \frac{\alpha}{|C_k|} \sum_{i \in C_k} w_{ik} - \frac{\beta}{|\overline{C_k}|} \sum_{i \in C_k} \overline{w_{ik}} \tag{4.10}$$

W przedstawionym wzorze C_k to zbiór dokumentów należących do kategorii k. Wartość w_{ik} to waga słowa i w dokumentach kategorii k. Wartość β modyfikuje wpływ dokumentów negatywnych. Najczęściej jako α przyjmuje się wartość 16, a jako β wartość 4.

Przypisywanie nowego dokumentu do kategorii odbywa się przez obliczenie tzw. kosinusowej miary podobieństwa wektora sprawdzanego dokumentu i wyliczonych wcześniej wektorów globalnych poszczególnych kategorii. Miara kosinusowa wylicza kosinus kąta miedzy dwoma wektorami. Klasyfikator przyporządkowuje dokument do kategorii o największej wartości kosinusa (najmniejszym kącie).

Przedstawiony opis pozwala sądzić, iż klasyfikacja wykonywana przy użyciu tego algorytmu jest bardzo szybka. Co więcej, większość przeprowadzanych przez nią obliczeń daje się przekształcić na odpowiednie zapytania SQL przez co doskonale nadaje się do implementacji z wykorzystaniem bazy danych. Dokonując pewnych drobnych zmian w reprezentacji danych można uzyskać wersję inkrementacyjną algorytmu uczącego.

4.4.3. Centroid

Klasyfikator Centroid [15] to w zasadzie pewne uproszczenie algorytmu Rocchio. Również tu powstaje jeden globalny wektor wag słów opisujący kategorię. W tym podejściu nadal stosowany jest system określania wag TFIDF opisany w poprzednim podrozdziale. W podobny sposób dokonuje się również mierzenia podobieństwa, stosując kosinus

kąta wektorów. Uproszczeniu ulega natomiast sposób tworzenia globalnego wektora kategorii na podstawie dokumentów treningowych. Nie stosuje się modyfikatorów wag. Wzór (4.11) przedstawia tworzenie wektora centroidalnego.

$$\mathbf{C} = \frac{1}{|S|}\sum_{\mathbf{d}\in S}\mathbf{d} \tag{4.11}$$

We wzorze **C** to powstały wektor centroidalny, *S* to zbiór wektorów dokumentów należących do kategorii, dla której tworzony jest globalny wektor, **d** to przygotowane wcześniej wektory poszczególnych dokumentów.

Rozwiązanie ma podobne wady i zalety implementacyjne jak klasyfikator Rocchio, ale dzięki prostszemu wzorowi tworzenia wektora globalnego daje się łatwiej przystosować do inkrementacyjnego uczenia.

4.4.4. KAN

Klasyfikator KAN [21] stara się uchwycić zależności między poszczególnymi słowami pojawiającymi się w dokumentach zbioru treningowego i dodatkowo wykorzystuje własny system miary podobieństwa stanowiący połączenie dwóch innych systemów.

Najpierw algorytm buduje sieć połączeń międzywyrazowych dla analizowanej kategorii, starając się odnaleźć połączenia dwuwyrazowe występujące w dokumentach treningowych więcej razy niż zadane minimum. Dodatkowo buduje standardowy wektor częstości wystąpień wyrazów.

W kolejnym kroku stara się określić tak zwaną wartość „przeświadczenia" (ang. *confidence*), czyli siłę wzajemnego związku znalezionych grup dwuwyrazowych. Innymi słowy, uzyskuje się wiedzę na temat tego, jaką można mieć pewność, iż wystąpienie jednego słowa pociągnie za sobą wystąpienie drugiego ze słów. Stosując wartość progową, można po uzyskaniu wartości przeświadczenia odfiltrować najmniej znaczące związki.

Algorytm wykorzystuje dodatkowo informację na temat między grupowych wystąpień słów w poszczególnych kategoriach. Jeśli słowo występuje tylko w jednej z kategorii, otrzymuje większą wartość wagi niż w przypadku, gdy w jednakowy sposób rozkłada się w wielu grupach.

Warto podkreślić, iż opisywany algorytm nie wylicza pełnej wagi słów w trakcie procesu uczenia. Wykonuje to zadanie w locie w momencie klasyfikacji nowego dokumentu, choć korzysta z informacji zebranych w fazie uczenia.

Klasyfikator KAN wykorzystuje miarę podobieństwa połączoną ze strategią progowania. Skupia się ona na odpowiednim doborze progu w przypadku zadania klasyfikacji wielokategoriowej. Z tego względu jej opis został w tej pracy pominięty.

Klasyfikator w oryginalnej wersji nie obsługuje inkrementacyjnego uczenia choć przez zastosowanie odpowiednich modyfikacji mógłby zostać do tego zadania przystosowany, tym bardziej, że część wartości wylicza dopiero w fazie klasyfikacji nowego dokumentu. Niestety algorytm ma złożoność obliczeniową $O(n^2)$ względem wymiaru przestrzeni cech, co oznacza, że nie nadaje się do obsługi przestrzeni cech (liczby słów stanowiących dane klasyfikatora) o znacznej wymiarowości.

4.4.5. DCM

Klasyfikator DCM [13] ma zgodnie z założeniami autorów spełnić kilka celów: zapewnić wysoką jakość klasyfikacji, niski koszt obliczeniowy, niskie wymagania pamięciowe związane z przechowywaniem danych, brak parametrów modyfikujących jego działanie i wysoką elastyczność w dostosowywaniu się do kategoryzacji różnych rodzajów dokumentów.

Klasyfikator ma pewne cechy wspólne z klasyfikatorem KAN a mianowicie również modyfikuje istotność słowa na podstawie jego występowania w wielu kategoriach. Pomniejsza znaczenie słowa, jeśli występuje ono w miarę jednorodnie we wszystkich kategoriach. Podobnie jak KAN pewne elementy ostatecznej wagi słów wylicza w locie na etapie klasyfikacji. Wykorzystuje mniej popularny sposób określania podobieństwa.

Algorytm wagę słowa w dokumencie wylicza inaczej niż w standardowych klasyfikatorach typu Rocchio, Centroid czy nawet k-NN. Przede wszystkim logarytmuje częstość wystąpienia słowa w dokumencie (nf_{id}) i przy jej liczeniu bierze pod uwagę długość dokumentu (l_d). W ten sposób słowo występujące w dokumencie czterokrotnie nie jest dwa razy ważniejsze od słowa występującego dwukrotnie. Wzór (4.12) przedstawia sposób wyliczania wagi słowa w dokumencie (w_{id}). Wartość ta jest uzyskiwana w fazie uczenia dla dokumentów treningowych i w fazie klasyfikacji dla kategoryzowanego dokumentu.

$$w_{id} = \frac{\log_2(nf_{id} + 1)}{\log_2(l_d + 1)} \tag{4.12}$$

Bardziej złożony wzór dotyczy wyliczania wagi słowa w kategorii (W_{ik}) i uwzględnia: relatywną wagę słowa w kategorii (WC_{ik}), relatywną istotność słowa we wszystkich

kategoriach (CC_i) i średnią istotność słowa w kategorii (AI_{ik}). Poniżej został przedstawiony wzór główny (4.13) wraz ze wzorami pomocniczymi (4.14), (4.15) i (4.16).

$$W_{ik} = AI_{ik}\left(\frac{WC_{ik}^2 \cdot CC_i^2}{\sqrt{WC_{ik}^2 + CC_i^2}} \cdot \sqrt{2}\right) \tag{4.13}$$

$$WC_{ik} = \frac{\log_2(df_{ik} + 1)}{\log_2(N_k + 1)} \tag{4.14}$$

$$CC_i = \log\frac{N \cdot \max_{k \in C_i}\{WC_{ik}\}}{\sum_{k=1}^{N} WC_{ik}} \cdot \frac{1}{\log N} \tag{4.15}$$

$$AI_{ik} = \left(\frac{\sum_{d \in k} w_{id}}{df_{ik}}\right)^{(2-WC_{ik})} \tag{4.16}$$

W przedstawionych wzorach N to łączna liczba kategorii obsługiwanych przez klasyfikator, df_{ik} to liczba dokumentów zawierających słowo i w kategorii k, C_i to zbiór kategorii zawierających słowo i. W wersji nieinkrementacyjnej algorytmu wagę słowa w kategorii można wyliczyć na etapie uczenia. Algorytm tworzy więc globalną informację o wagach słów w kategoriach, czym przypomina klasyfikatory Rocchio i Centroid.

DCM do określania podobieństwa dokumentu do kategorii wykorzystuje współczynnik Jaccarda zamiast iloczynu wektorowego lub kosinusa. Wzór (4.17) dotyczy wyliczania podobieństwa (S_{dk}). Warto podkreślić, iż w trakcie jego wyliczania pod uwagę brane są tylko słowa występujące w analizowanym dokumencie.

$$S_{dk} = \frac{\sum_{i \in d}(w_{id} \cdot W_{ik})}{\sum_{i \in d} w_{id}^2 + \sum_{i \in d} W_{ik}^2 - \sum_{i \in d}(w_{id} \cdot W_{ik})} \tag{4.17}$$

Algorytm klasyfikacji posiada dwie wersje: inkrementacyjną i zwykłą. Z przedstawionego opisu wynika również, iż jego implementacja z wykorzystaniem bazy danych nie powinna być trudna, szczególnie jeśli język SQL stosowanej bazy zawiera rozszerzenia pozwalające wyliczać logarytmy.

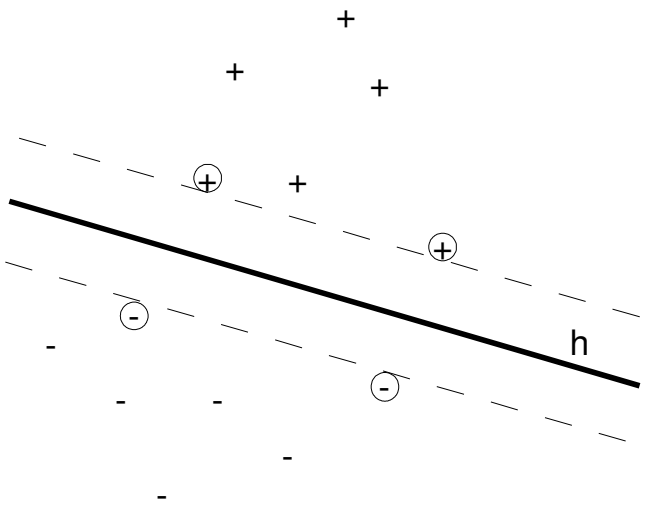
4.4.6. SVM

Klasyfikator SVM (*Support Vector Machines*, [18]) został zaprojektowany, by doskonale radzić sobie nawet z bardzo dużą liczbą słów przy jednoczesnym zachowaniu

krótkiego czasu klasyfikacji. Standardowo dokonuje klasyfikacji binarnej i zakłada liniową separację przykładów pozytywnych i negatywnych.

Podstawowa wersja klasyfikatora do wstępnego określania wag słów stosuje wzór TFIDF. Następnie używa algorytmu wyboru słów istotnych, by w ten sposób zmniejszyć ogólną wymiarowość wektora słów.

Właściwy klasyfikator bazuje na zasadzie minimalizacji ryzyka strukturalnego wywodzącej się z teorii uczenia statystycznego. Ogólnie można powiedzieć, że w trakcie uczenia SVM stara się wygenerować hiperpłaszczyznę decyzyjną, która maksymalizuje margines między pozytywnymi i negatywnymi przykładami dokumentów ze zbioru treningowego. Rysunek 4.1 obrazuje tworzenie takiego rozdzielenia.



Rysunek 4.1. Hiperpłaszczyzna w klasyfikatorze SVM

W trakcie nauki klasyfikator wymaga minimalizacji funkcji, co w tym przypadku jest zadaniem związanym z programowaniem kwadratowym (QP, ang. *Quadratic Programming*). Początkowo zadanie programowania kwadratowego było bardzo złożone obliczeniowo — proporcjonalne do kwadratu liczby dokumentów treningowych. W ostatnim czasie powstały algorytmy zapewniające podział głównego zadania na kilka podzadań, obliczania ich osobno a następnie łączenia, co znacząco podniosło szybkość uczenia.

Algorytm stosowany do rozwiązywania przypadków z separacją liniową można rozszerzyć o obsługę separacji nieliniowej, wprowadzając hiperpłaszczyzny o łagodnych granicach lub też przekształcając oryginalne wektory do wyższych wymiarów.

Klasyfikator SVM po dokonaniu w algorytmie uczącym modyfikacji związanych z inną obsługą QP mógłby obsługiwać uczenie inkrementacyjne. Proces klasyfikacji bardzo dobrze nadaje się do wykorzystania z bazą danych. Nie można niestety powiedzieć tego samego o fazie uczenia.

4.4.7. Winnow

Klasyfikator Winnow [11], [32] to algorytm statystyczny, który nie korzysta z obliczania prawdopodobieństw jak Naive Bayes. Zamiast tego wylicza dla każdej klasy wynik. Stanowi przykład wykorzystania rozwiązań znanych z tradycyjnych sieci neuronowych w zadaniach klasyfikacji tekstów. Istnieje kilka wersji klasyfikatora: Positive Winnow, Perceptron i Balanced Winnow. W tym podrozdziale zostanie omówiony ostatni z nich z racji jego największej skuteczności.

Algorytm uczący stosuje dowolny system wyliczania wag poszczególnych słów dokumentu przedstawianego do klasyfikacji. Wersja Balanced dla każdego słowa przechowuje dwie wagi w^+ i w^-. Rzeczywisty współczynnik słowa stanowi różnicę między nimi. Oznacza to, że mogą wystąpić współczynniki ujemne. Dla klasyfikowanego dokumentu z wagami słów $d = (d_1, d_2, d_3, ..., d_n)$ algorytm przewiduje przypisanie dokumentu do testowanej kategorii wtedy, gdy spełniona zostanie nierówność podana wzorem (4.18).

$$\sum_{i=1}^{n}(w_i^+ - w_i^-)d_i > \theta \tag{4.18}$$

Wartość θ to przyjęty dla danej kategorii próg aktywacji. Za każdym razem analizowane są tylko słowa zawarte w klasyfikowanym dokumencie.

Początkowo wagę w^+ słów inicjalizuje się dwukrotną wartością stosowanego progu θ natomiast wagę w^- wartością progu. W trakcie modyfikacji algorytm stosuje dwa mnożniki: promocji α i degradacji β. Pierwszy z nich ustawiony jest na wartość większą od 1 ($\alpha > 1$), najczęściej wynoszącą 1,33; 1,5 lub 2. Współczynnik degradacji ustawiony jest na wartość w przedziale od 0 do 1 ($0 < \beta < 1$), najczęściej wynoszącą 0,75; 0,66 lub 0,5.

Algorytm uczy się tylko w sytuacji, gdy dokona błędu w klasyfikacji dokumentu. Modyfikuje wtedy wagi związane ze słowami biorącymi udział w klasyfikacji. Jeśli dokument uczący był pozytywny i został źle zaklasyfikowany, wartość wagi w^+ zostaje pomnożona przez α natomiast wartość wagi w^- pomnożona przez β, co zwiększa ogólny współczynnik słowa. Jeśli dokument był negatywny i został źle sklasyfikowany, wartość wagi

w^+ zostaje pomnożona przez β natomiast wartość wagi w^- pomnożona przez α, co zmniejsza ogólny współczynnik słowa.

Wykazano, że ten klasyfikator potrafi w efektywny sposób nauczyć się dowolnej funkcji o liniowym progu i radzi sobie stosunkowo dobrze, gdy nie istnieje liniowa separacja obszarów. Analizy teoretyczne wykazały, iż algorytm zachowuje się poprawnie nawet w przypadku szumu informacyjnego i nieistotnych słów. Dodatkowo umożliwia dryft sposobu separacji (aproksymowanej funkcji) dzięki swej inkrementacyjnej naturze.

Algorytm doskonale nadaje się do zastosowania z bazami danych, gdyż nie wymaga przeprowadzania złożonych obliczeń. Co więcej, ma wbudowaną obsługę inkrementacyjnego uczenia — modyfikacja wag może następować w dowolnym momencie po niepoprawnej klasyfikacji dokumentu.

4.4.8. Widrow-Hoff

Klasyfikator Widrow-Hoff [21] jest w dużej mierze podobny do algorytmu Winnow. Również zapewnia inkrementacyjne uczenie w przypadku napotkania błędu lub do osiągnięcia wartości błędu poniżej pewnej liczby. Jest też stosunkowo łatwy w implementacji bazodanowej. Algorytm wykorzystuje dla każdej kategorii jeden wektor słów z wagami i stosuje sposób określania kategorii jak w Winnow. Do modyfikacji wag używa wzoru (4.19).

$$y_{ij} = y_{ij-1} - 2\alpha(\mathbf{w}_{j-1} \cdot \mathbf{d}_j - b_j)x_{ij} \tag{4.19}$$

W przedstawionym wzorze y_{ij} to nowa waga słowa w wektorze $\mathbf{w}_j$, x_{ij} to waga w wektorze d_j, $\mathbf{w}_{j-1}$ to stary wektor cech, $\mathbf{d}_j$ to nowy dokument treningowy, α to parametr szybkości uczenia (wartość znacznie mniejsza od jedności), b_j to oznacznik pozytywny-negatywny (wynosi 1, gdy dokument jest pozytywny lub 0, gdy stanowi przykład negatywny). Początkowy wektor wag zawiera najczęściej same zera (ewentualnie mogą to być niewielkie wartości losowe znacznie mniejsze od jedności).

4.5. KLASYFIKATORY BAZUJĄCE NA PRZYKŁADACH

Klasyfikatory bazujące na przykładach nie tworzą jawnej, uogólnionej reprezentacji kategorii dokumentów, ale dokonują klasyfikacji porównując nowy dokument z tymi, które zostały im przekazane w trakcie uczenia. Z tego powodu nazywa się je klasyfikatorami leniwymi, gdyż nie stosują rzeczywistej fazy nauki. W tym podrozdziale przedstawione zostaną dwa klasyfikatory wykorzystujące tę zasadę k-NN i bazujący na nim WAKNN.

4.5.1. k-NN

Algorytm k-NN (k najbliższych sąsiadów) [21], [30] nie stosuje w ogóle fazy uczenia. Zapamiętuje jedynie wszystkie dokumenty treningowe w postaci wektorów wag. Takie podejście motywuję się tym, że sam dokument ma większą siłę reprezentacyjną niż jego uogólniony profil stosowany w wielu innych klasyfikatorach. Co więcej, podejście zapewnia dobrą jakość działania w przypadku braku liniowej separacji dokumentów.

Klasyfikacja nowego dokumentu polega na wyliczeniu jego podobieństwa do wszystkich zapamiętanych dokumentów treningowych. Następnie zebrane wyniki podobieństwa sortuje malejąco. Do dalszej analizy branych jest *k* najlepszych wyników. Po ograniczeniu wyników dzieli się je względem kategorii, do których przynależą dokumenty treningowe i z których były generowane. Wyniki poszczególnych kategorii sumuje się. W ten sposób powstaje łączny wynik podobieństwa dla poszczególnych kategorii. W zwykłym rozwiązaniu dwukategoriowym nowy dokument przypisuje się do tej kategorii, która uzyskała wyższy wynik.

Algorytm wykorzystuje różne miary podobieństwa — obecnie najpopularniejszą z niej jest kosinus kąta wektorów, ale dawniej stosowano inne podejścia.

Klasyfikacja dokumentów w ten sposób choć niejednokrotnie skuteczna, ma kilka wad. Po pierwsze, trzeba określić w jakiś sposób najbardziej optymalną wartość *k*. Duża wartość może działać dobrze dla dużych kategorii ale dyskryminować mniejsze o niewielkiej liczbie przykładów pozytywnych. Na błędną kategoryzację mogą mieć wpływ również dokumenty treningowe niepoprawnie przypisane do kategorii przez ekspertów dziedzinowych (w rozwiązaniach tworzących globalny wektor jest to częściowo znoszone). Szybkość działania tego algorytmu klasyfikacji jest proporcjonalna do liczby dokumentów treningowych, gdyż dla każdego z nich musi zostać policzone podobieństwo względem analizowanego dokumentu.

Algorytm k-NN dobrze nadaje się do inkrementacyjnego uzupełniania bazy wiedzy, gdyż nie stosuje wyrafinowanej fazy uczącej. Jego implementacja z wykorzystaniem bazy danych nie powinna sprawiać żadnych problemów, ponieważ wymaga jedynie obliczania podobieństw.

4.5.2. WAKNN

Klasyfikator WAKNN [16] to algorytm bazujący na podstawowej wersji k-NN, ale wprowadzający fazę uczenia. Polega ona na modyfikacji wag słów zapamiętanych doku-

mentów treningowych. W każdym kroku iteracyjnym wagi każdego ze słów są modyfikowane o pewną niewielką wartość w celu sprawdzenia, czy zmiana ta przynosi poprawę klasyfikacji. Algorytm wybiera te słowa, dla których modyfikacja wag przyniosła znaczącą poprawę w reprezentacji kategorii. Następnie zapamiętuje nowe wagi tych słów. Cały proces powtarza się wielokrotnie.

Przedstawione podejście ma wiele wad zwykłego rozwiązania k-NN, czyli przede wszystkim złożoność obliczeniową zależną od liczby dokumentów treningowych. Co gorsza wprowadza znaczący czas związany z uczeniem — złożoność obliczeniowa tej fazy wynosi $O(cn^2)$ a więc jest ogromna. Choć podejście to nadaje się do implementacji bazodanowej, mogą pojawić się problemy z zapewnieniem inkrementacyjności uczenia (dodanie nowego dokumentu może zachwiać wcześniej zoptymalizowanymi wagami). Podsumowując, algorytm ten jest propozycją ulepszenia k-NN, ale wiąże się z ogromnym kosztem obliczeniowym rosnącym znacząco wraz z przybywaniem kolejnych dokumentów treningowych.

4.6. KLASYFIKATORY ZŁOŻONE

Ostatnia grupa klasyfikatorów to algorytmy korzystające z wielu faz niezależnych klasyfikacji. Dodatkowo w tej grupie znalazł się również algorytm starający się połączyć zalety klasyfikatorów Rocchio i k-NN.

4.6.1. Klasyfikator komisyjny

Rozwiązanie komisyjne [30] wykorzystuje spostrzeżenie, iż kilku ekspertów dziedzinowych głosujących nad przyznaniem kategorii nowemu dokumentowi uzyskuje lepsze wyniki niż pojedynczy ekspert wykonujący to samo zadanie. Postanowiono przenieść to podejście do automatycznej klasyfikacji tekstów.

System komisyjny korzysta z kilku najczęściej całkowicie od siebie niezależnych klasyfikatorów. Dokument do klasyfikacji jest analizowany przez wszystkie klasyfikatory składowe. Przyznawanie końcowej kategorii dokumentu może polegać na wybraniu tej kategorii, która została wskazana przez większość klasyfikatorów lub na wybraniu wartości tego klasyfikatora, który zapewnił największą różnicę między wybraną kategorią a innymi (czyli jest najbardziej pewien swego wyboru).

Podejście to ma wadę związaną ze znaczną złożonością wynikającą z potrzeby korzystania jednocześnie z kilku systemów klasyfikacji. Łatwość implementacji bazodanowej i uczenia inkrementacyjnego uzależniona jest od właściwości użytych algorytmów.

4.6.2. Boosting

Boosting [5] to stosunkowo nowa koncepcja, która w dosyć szczególny sposób traktuje pojęcie komisyjnego ustalania kategorii dla dokumentu. Stosuje pewną liczbę klasyfikatorów tego samego typu, które w pojedynkę mogą działać tylko odrobinę lepiej od zwykłego zgadywania, łącząc je w jeden dobrej jakości klasyfikator. Ciekawość tego rozwiązania polega na tym, iż poszczególne klasyfikatory nie są uczone w sposób niezależny (równolegle), ale sekwencyjnie (jeden po drugim). W ten sposób kolejny klasyfikator w ciągu może koncentrować się na tych dokumentach, które sprawiły problemy poprzednim klasyfikatorom.

Najczęściej jako klasyfikatory składowe w algorytmie Boosting wykorzystuje się bardzo proste rozwiązania o dużej szybkości działania. Przykład takiego klasyfikatora przedstawia wzór (4.20).

$$s(x) = \begin{cases} c_{0i} & if\ x_i = 0 \\ c_{1i} & if\ x_i = 1 \end{cases} \tag{4.20}$$

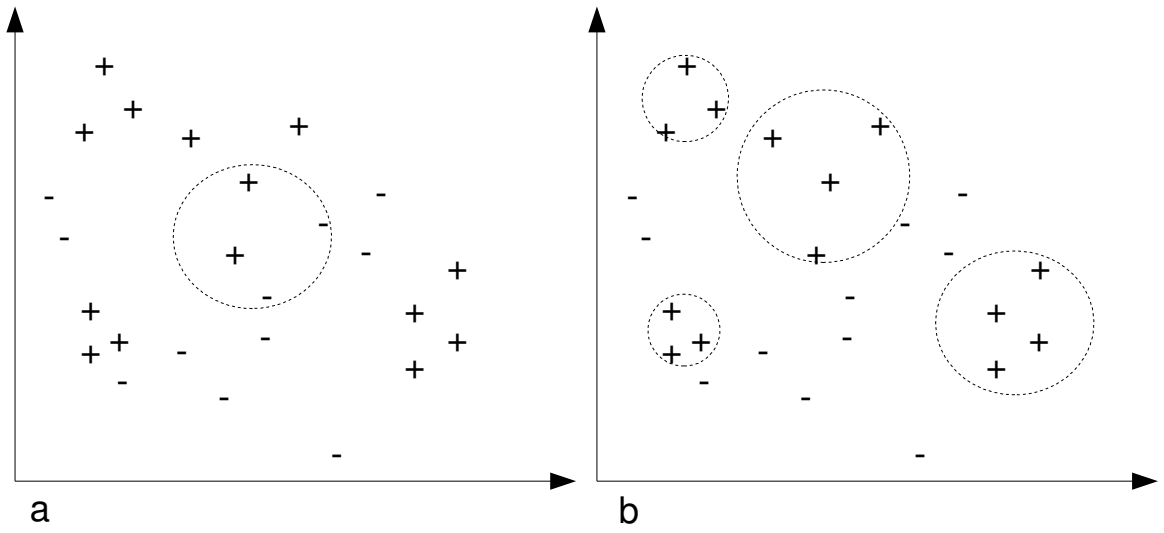
W przedstawionym wzorze s to funkcja używana do klasyfikacji, c_{0i} i c_{1i} to wartości przyporządkowywane poszczególnym słowom (wagi). Zmieniają się one w kolejnych klasyfikatorach ciągu, by minimalizować błąd wprowadzany przez poprzednie klasyfikatory. Ich określenie następuje na etapie uczenia. Wartości w_i to wagi testowanego dokumentu (1 oznacza wystąpienie słowa, 0 jego brak).

Klasyfikator tego rodzaju stosunkowo słabo nadaje się do szybkiej implementacji z wykorzystaniem bazy danych. Nie zapewnia też wygodnej metody uczenia inkrementacyjnego.

4.6.3. kNN (Rocchio + k-NN)

Klasyfikator kNN [14] bazuje na pomyśle zaczerpniętym z innej propozycji nazwanej GIS (*Generalized Instance Set*). Opiera się on na połączeniu zalet klasyfikatorów Rocchio (duża szybkość, odporność na szum, tworzenie globalnych profili kategorii) z zaletami klasyfikatora k-NN (obsługa separacji nieliniowej, dobra jakość klasyfikacji) przy jednoczesnym zniesieniu wad obu rozwiązań.

Rysunek 4.2 przedstawia w jaki sposób algorytm stara się poprawić jakość klasyfikacji, gdy dane nie są łatwo separowalne liniowo, w porównaniu z uogólnieniem kategorii proponowanym przez algorytm Rocchio.



Rysunek 4.2. Reprezentacja kategorii + w klasyfikatorze Rocchio (a) i kNN (b)

Przedstawiony rysunek wykorzystuje odległość euklidesową jako miarę podobieństwa, choć w rzeczywistym systemie jest to podobieństwo kosinusowe. Utworzony przez algorytm Rocchio ogólny profil kategorii powoduje przy braku dobrej separacji znacznych błędów. W algorytmie k-NN błąd byłby minimalny, ale należałoby sprawdzić wszystkie dokumenty treningowe. Oznaczałoby to kilkunastokrotnie większy czas klasyfikacji niż w przypadku algorytmu Rocchio. Algorytm kNN stara się utworzyć wiele ogólnych profili dla podobnych do siebie dokumentów, dzieląc przestrzeń według podobieństwa. W ten sposób zamiast porównywać kilkanaście dokumentów, w trakcie klasyfikacji używa się kilku uogólnionych profili podobnych dokumentów. Czas klasyfikacji zmniejsza się przy jednoczesnym uzyskaniu dobrej jakości pomimo braku separacji liniowej.

Algorytm konstrukcji wielu grup uogólniających tworzy w trakcie uczenia macierz podobieństw poszczególnych dokumentów. Następnie wszystkie dokumenty określa jako niezgrupowane i na podstawie macierzy wyznacza lokalnych sąsiadów każdego dokumentu. Po tej operacji wybiera niezgrupowany dokument z największą liczbą sąsiadów o dobrym podobieństwie. Tworzy na podstawie tego zlepka dokumentów jego profil uogólniony i wylicza podobieństwo nowego uogólnienia do dokumentów do niego należących. Najsłabsze z tych podobieństw oraz liczba dokumentów zlepka zostaje zapamiętana. Dokumenty, dla których powstało uogólnienie algorytm zaznacza jako zgrupowane. Cały proces

powtarza się aż do momentu, w którym nie będzie żadnych nieoznaczonych dokumentów treningowych. Na końcu algorytm generuje ogólny profil ze wszystkich dokumentów (jak w Rocchio).

Klasyfikacja nowego dokumentu odbywa się w następujący sposób. Obliczone zostaje podobieństwo do wszystkich uogólnionych profili. Pod uwagę w dalszej analizie brane są jednak tylko te profile, dla których podobieństwo było większe od zapamiętanego podobieństwa minimalnego. Dokument zostaje przypisany do tej kategorii, która uzyskała największą sumę podobieństw.

Rozwiązanie to można dostosować do nauki inkrementacyjnej, modyfikując algorytm uczący. Algorytm dobrze nadaje się do implementacji z wykorzystaniem bazy danych.

4.7. PORÓWNANIE KLASYFIKATORÓW I WYBÓR ROZWIĄZAŃ O ODPOWIEDNICH WŁAŚCIWOŚCIACH

Porównanie wszystkich przedstawionych w tym rozdziale klasyfikatorów ze względu na jakość klasyfikacji, czas działania, łatwość implementacji bazodanowej i inkrementacyjny proces nauki nie jest zadaniem łatwym. Autorzy publikacji stosują kilka różnych korpusów testowych (najpopularniejsze to Routers-21578, 20-Newsgroups i OHSUMED; wszystkie dotyczą tekstów w języku angielskim), różne miary oceny jakości (F_1, punkt przełamania) i niejednokrotnie wprowadzają własne modyfikacje do podstawowych wersji algorytmów. Z tych powodów nie zdecydowano się na wykonanie pełnego porównania wszystkich algorytmów w prostej tabeli, ale przedstawienie wyników kilku porównań zawierających niektóre klasyfikatory. Każde z porównań było przeprowadzane przez autorów publikacji, więc można na ich podstawie wysnuć wnioski co do względnej jakości użytych klasyfikatorów.

Znacznie trudniej uzyskać miarodajne dane na temat szybkości działania klasyfikatorów. Podaje je niewiele prac — w jednych są to szczegółowe informacje, w niektórych tylko krótkie zdania typu „A był szybszy od B". Poza tym prace podają najczęściej tylko czasy związane z klasyfikacją a pomijają informację o czasach uczenia klasyfikatora.

4.7.1. Ocena jakości klasyfikacji

Tabela 4.2 zawiera porównania jakości klasyfikacji zaczerpnięte z publikacji i raportów technicznych udostępnianych w internecie. Puste pola oznaczają, że w danym rozwiązaniu autorzy nie porównywali tego klasyfikatora. Nagłówki kolumn zawierają liczby,

gdyż pełny opis sposobu uzyskania porównania w tym miejscu zaciemniłby obraz klasyfikacji. Znaczenie liczb zostało wyjaśnione pod tabelą.

Tabela 4.2. Porównanie jakości klasyfikatorów zebrane z różnych publikacji. Wszystkie wartości podane w procentach

Porównanie / **Klasyfikator**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Drzewa decyzyjne			84,5	67,0		79,4		77,9	
Naive Bayes			91,0	65,0		72,0	54,4	80,5	
Rocchio	85,1	78,8		66,0	57,3	79,9			
Centroid								80,4	
KAN					62,9				
DCM							72,0		
SVM	91,5	83,6				86,4	71,2		
Balanced Winnow				74,7					
Widrow-Hoff					51,7				
k-NN	83,2	79,1	90,6		57,8	82,3		78,9	
WAKNN			90,0						
Boosting									72,7
kNN	86,1	80,8							

1 — Korpus: Routers-21578, makro F_1, [14]

2 — Korpus: 20-newsgroups, makro F_1, [14]

3 — Korpus: Routers-21578, dokładność, [16]

4 — Korpus: Routers-21578 (podział Lewisa), punkt przełamania, [11]

5 — Korpus: Routers-21578 (najlepsze wyniki dla miary *RinSCut*), makro F_1, [21]

6 — Korpus: Routers-21578, mikro dla punktu przełamania, [18]

7 — Korpus: Routers-21578, makro F_1, [13]

8 — Korpus: Routers-21578, dokładność?, [15]

9 — Korpus: Routers-21578, makro F_1, [5]

Analizując przedstawione porównanie można dojść do wniosku, iż jako klasyfikatory odniesienia najczęściej stosowane są klasyfikatory Rocchio, k-NN i Naive Bayes. Jakość działania tych trzech klasyfikatorów jest bardzo podobna: raz wygrywa jeden z nich, innym razem drugi, ale ogólne wyniki są zbliżone (różnice nie przekraczają 2%). Klasyfikator stosujący drzewa decyzyjne okazał się bardzo nierówny — raz okazywał się lepszy od klasyfikatorów z głównej trójki, raz gorszy. Może to wynikać ze szczegółów implementa-

cji lub innego rozkładu danych uczących i testowych. Klasyfikator Widrow-Hoff nie najlepiej radził sobie, będąc najgorszy w przytoczonym porównaniu. Klasyfikator Centroid okazał się podobny (a w nieprzedstawionych testach odrobinę lepszy od klasyfikatorów k-NN i Bayes), co nie powinno dziwić z racji dużego podobieństwa do Rocchio. Algorytm kNN w obu testach był lepszy od podstawowych klasyfikatorów, ale ulegał w istotny sposób klasyfikatorowi SVM. Klasyfikator WAKNN pomimo swojej dużej złożoności obliczeniowej nie potrafił znacząco wyprzedzić zwykłego k-NN. Klasyfikator KAN choć radził sobie lepiej od swoich konkurentów w przedstawionym porównaniu to jednak nie na tyle, by zrekompensować jego znaczną złożoność obliczeniową dla wielu słów. Algorytm Balanced Winnow okazywał się lepszy w swoi porównaniu od konkurentów i to znacząco (8%). Duże trudności z porównaniem dotyczą rozwiązania Boosting, gdyż nie udało się dotrzeć do żadnego testu porównawczego. Przedstawiona wartość dotyczy dokładnych testów rozwiązania AdaBoost, ale nie znalazły się niej żadne odniesienia do innych klasyfikatorów. Z testów wynika, że najlepszym klasyfikatorem w kilku porównaniach okazał się SVM. W teście porównawczym DCM okazał się równie dobry jak SVM.

Podsumowując, najlepszymi klasyfikatorami w połączonych rozwiązaniach okazały się DCM i SVM. W wielu innych pracach SVM również został uznany za jeden z najlepszych klasyfikatorów. DCM był porównywany tylko w jednej pracy i z niewiadomych powodów nie zyskał dużej popularności. Kolejnym klasyfikatorem osiągającym dobre wyniki na tle konkurencji jest Balanced Winnow. Dobre wartości uzyskiwały w porównaniu z konkurencją również kNN i KAN, choć nie były one tak spektakularne.

4.7.2. Ocena szybkości działania

Szczegółowe dane na temat szybkości działania zostały odnalezione tylko w dwóch pracach [13], [14]. Co gorsza, tylko w jednej z nich podano czas związany z uczeniem klasyfikatora. Prace przedstawiają czas związany z algorytmami: SVM, k-NN, Rocchio, kNN, DCM i Naive Bayes. Na podstawie tych szczątkowych danych można było wyciągnąć pewne wnioski.

Najwolniejszym algorytmem w czasie klasyfikacji okazał się k-NN. Nie powinno to dziwić z racji proporcjonalności czasu jego działania do liczby dokumentów treningowych (im więcej, tym czas się wydłuża). Drugim algorytmem o długim czasie działania okazał się DCM (warto tu jednak podkreślić, iż czas ten nie jest uzależniony od liczby dokumentów treningowych). Algorytmy SVM i kNN okazują się znaczącą szybsze od poprzednich

dwóch (można powiedzieć, że 2-3 krotnie). Algorytmy Rocchio i Naive Bayes charakteryzowały się największą szybkością (około 1,5 razy szybsze od SVM).

Czasy uczenia można było porównać tylko dla trzech klasyfikatorów: SVM, Naive Bayes i DCM. Najszybszy w tej realizacji okazywał się Naive Bayes. Niewiele za nim uplasował się DCM. SVM w trakcie uczenia okazał się najwolniejszy (rósł bardzo szybko wraz ze wzrostem liczby dokumentów treningowych).

4.7.3. Wybór klasyfikatorów do implementacji

Do implementacji zdecydowano się wybrać cztery algorytmy klasyfikacji tekstów: DCM+ (inkrementacyjna wersja DCM), Balanced Winnow i k-NN. Czwarty klasyfikator będzie rozwinięciem kNN o uczenie inkrementacyjne z pewnymi dodatkami zapożyczonymi z DCM. Oto powody takiego wyboru.

W przedstawionych porównaniach klasyfikator DCM spisywał się równie dobrze jak SVM. Jego wybór zamiast SVM spowodowany był łatwością implementacji z wykorzystaniem bazy danych oraz prostym sposobem obsługi uczenia inkrementacyjnego.

Klasyfikator Balanced Winnow został wybrany z powodu swej prostoty (co powinno pozwolić mu uzyskiwać szybkość porównywalną z algorytmami Rocchcio i Naive Bayes), która jednak daje mu górować nad wspomnianymi algorytmami, wbudowanej obsługi uczenia inkrementacyjnego i łatwości implementacji z zastosowaniem bazy danych.

Komentarze niekoniecznie muszą zapewniać łatwą do przeprowadzenia separację liniową. Algorytm k-NN choć znacznie wolniejszy i niejednokrotnie mniej dokładny może w takiej sytuacji okazać się skuteczny o ile dokona się drobnych modyfikacji w celu częściowego uchronienia go przed szumem. Klasyfikator k-NN doskonale nadaje się do implementacji bazodanowej i zapewnia inkrementacyjne uczenie.

Ostatni z klasyfikatorów również motywowany jest nieliniowością podziału komentarzy. Będzie to własne rozwiązanie starające się wcielić w życie pomysł kNN, ale z zapewnieniem inkrementacyjnego sposobu uczenia i zastosowaniem innej miary podobieństwa. Rozwiązanie to również nadaje się do wykorzystania z relacyjną bazą danych, choć nie tak dobrze jako pozostałe. Wynika to z potrzeby stosowania bardziej wyrafinowanej logiki do odnajdywania i łączenia grup podobnych komentarzy.

5. IMPLEMENTACJA ISTNIEJĄCYCH ALGORYTMÓW KLASYFIKACJI I NOWA PROPOZYCJA

5.1. OGÓLNE WARUNKI IMPLEMENTACYJNE

5.1.1. Podstawowe dane

Wszystkie algorytmy zostały zaimplementowane w języku skryptowym PHP 5.1 [2] i korzystają z relacyjnej bazy danych MySQL 5 [23]. Wszystkie korzystają z tego samego algorytmu preparacji danych opisanego w rozdziale 3. i są podklasami jednej abstrakcyjnej klasy klasyfikatora (`Classify`). W ten sposób istnieje możliwość łatwej podmiany ich w dowolnym rozwiązaniu.

Każda klasa konkretnego klasyfikatora znajduje się w osobnym pliku i udostępnia trzy metody publiczne zapewniające:

- wstępne uczenie klasyfikatora po podaniu dowolnej liczby przykładów pozytywnych i negatywnych (metoda `doInit()`),
- dokonywanie właściwej klasyfikacji po podaniu dokumentu do sprawdzenia (metoda `doClassify()`),
- inkrementacyjne uczenia klasyfikatora po określeniu typu dokumentu i jego treści (metoda `doUpdate()`).

Ponieważ weryfikacja komentarzy jest zadaniem dwuklasowym (zatwierdź-odrzuć), o ile tylko to było możliwe stosowano optymalizacje wykorzystujące ten fakt. Z racji dużej powolności języka skryptowego PHP część zadań obliczeniowych nie wymagających wcześniejszego pobierania starano się przenieść do bazy danych, wykorzystując odpowiednie aktualizacyjne zapytania SQL.

Kolejne podrozdziały zawierają szczegółowe opisy sposobu implementacji klasyfikatorów. Jest to o tyle istotne, iż niewiele prac związanych z klasyfikatorami wykorzystuje do przechowywania zwykle relacyjne bazy danych a sam kod implementuje w powolnym języku skryptowym.

5.1.2. Odrzucanie niepewnych

Wszystkie przedstawione algorytmy zawierają dodatkowy element ujawniający się w trakcie wykonywania klasyfikacji nowych komentarzy. W tradycyjnym rozwiązaniu kategoryzacji binarnej wygrywa ta kategoria, która osiągnie większą wartość podobień-

stwa. Aby uzyskać mniejszą stopę błędów i możliwość rozstrzygania niejasności przez moderatora, klasyfikator odmawia wskazania kategorii dokumentu, jeśli różnica podobieństw jest mniejsza od zadanego progu.

W fazie testów progi dla poszczególnych klasyfikatorów i zbiorów komentarzy były określane empirycznie. Przedstawione rozwiązanie jest elastyczne, gdyż moderator sam może zdecydować o wartości ewentualnego błędu, zmniejszając lub zwiększając wartość progu. Dla progu wynoszącego 0 zastosowane podejście działa tak samo, jak rozwiązanie standardowe.

5.2. ALGORYTM BALANCED WINNOW

5.2.1. Organizacja tabel bazy danych

Rysunek 5.1 przedstawia zawartość dwóch tabel wykorzystywanych przez klasyfikator do przechowywania danych. Tabela kończąca się na `n` dotyczy komentarzy negatywnych natomiast tabela dotycząca komentarzy pozytywnych kończy się literą `p`. Tabele zawierają jedynie słowa oraz dotyczące ich wagi pozytywne i negatywne. Tabele są więc równoważne wektorowi cech (słów) w innych systemach przechowywania danych.

Rysunek 5.1. Schemat tabel klasyfikatora Balanced Winnow

5.2.2. Uczenie wstępne

Uczenie wstępne na samym początku czyści tabele bazy danych, by rozpocząć wszystko od nowa. Następnie konwertuje otrzymane komentarze treningowe z postaci oryginalnej na wektory słów z częstościami. Dodatkowo w tym samym czasie kreuje osobną tablicę z wszystkimi słowami z dokumentów (wykonuje osobne listy dla komentarzy pozytywnych i negatywnych).

Znajomość liczby słów pozwala określić początkowe wartości wag. W tradycyjnym rozwiązaniu wagi początkowe mają wartość proporcjonalną do stosowanego progu. W klasyfikacji binarnej nie stosuje się progu i nie używa go wykonywany algorytm (wygrywa kategoria o większej wartości). Z tego powodu początkowe wagi są proporcjonalne do

odwrotności liczby słów w kategorii przy czym waga pozytywna uzyskuje wartość dwukrotnie większą od wagi negatywnej.

Wyrazy z początkowymi wagami zostają umieszczone w tabelach bazy danych po czym algorytm przechodzi do fazy uczenia wstępnego. W uczeniu tym każdy komentarz jest oceniany w pętli. Jeśli nie zostanie poprawnie zaklasyfikowany lub też nie zostanie przekroczona określona liczba iteracji, dochodzi do aktualizacji wag w sposób podany w podrozdziale 4.4.7 i ponownego wejścia do pętli. Ten sam krok jest przeprowadzany osobno dla komentarzy pozytywnych i negatywnych. Na tym kończy się uczenie wstępne.

5.2.3. Klasyfikacja

Metoda klasyfikująca przetwarza komentarz z wersji oryginalnej na wektor. Jeśli na tym etapie zostanie zwrócona wartość `false`, oznacza to, że komentarz zawiera wulgaryzmy, adres URL lub e-mail, albo jest pusty. Metoda dla takiego przypadku od razu przypisuje komentarz do kategorii negatywnej.

Kolejny etap to określenie oceny dokumentu dla kategorii pozytywnej i negatywnej. Wyliczenia różnicy wag dokonuje baza danych dzięki odpowiedniemu zapytaniu SQL. Dodatkowo wydobywane są tylko te słowa, które występują w analizowanym dokumencie. Dla każdego zwróconego przez bazę słowa algorytm wylicza ostateczną wagę i dodaje ją do całościowej sumy wag. W porównaniu z oryginalnym Balanced Winnow zaszła drobna modyfikacja. Wynikowa waga jest przed dodaniem do sumy wag pierwiastkowana. Oryginalny wzór z podrozdziału 4.4.7 zmienia się we wzór (5.1).

$$\text{sum} = \sum_{i=1}^{n} \sqrt{(w_i^+ - w_i^-)d_i} \tag{5.1}$$

Po uzyskaniu łącznych sum algorytm sprawdza, czy któraś z nich jest większa od drugiej o co najmniej próg niepewności. Jeśli tak, zwraca informację o określeniu komentarza jako pozytywnego lub negatywnego. W przeciwnym razie informuje o braku rozstrzygnięcia co do rodzaju komentarza.

5.2.4. Uczenie inkrementacyjne

Metoda ucząca przetwarza komentarz z wersji oryginalnej na wektor. Następnie przystępuje do aktualizacji wag zawartych w komentarzu słów w sposób opisany w podrozdziale 4.4.7. Jedyna różnica polega na tym, iż zawsze aktualizuje obie kategorie. Przypuśćmy, że komentarz został błędnie zaklasyfikowany jako poprawny. Algorytm postara

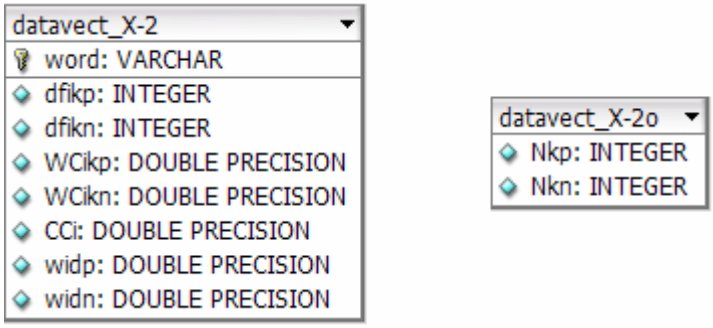
się zmniejszyć wagi słów tego komentarza z zbiorze słów kategorii pozytywnej i zwiększyć je dla tych samych słów ze zbioru kategorii negatywnej.

Po każdej aktualizacji algorytm usuwa ze zbiorów słów te, które mają bardzo małą lub bardzo dużą wartość wagi. Testy wykazały, że pozwala to pozbyć się słów, które nie uczestniczą znacząco w klasyfikacji lub też ją zaburzają, a nie wpływa negatywnie na jakość klasyfikacji. Mniejsza liczba słów przy kolejnych klasyfikacjach oznacza szybszy czas działania.

5.3. ALGORYTM DCM+

5.3.1. Organizacja tabel bazy danych

Rysunek 5.2 przedstawia zawartość dwóch tabel wykorzystywanych przez klasyfikator do przechowywania danych. Tabela kończąca się na `o` jest tabelą zawierającą zawsze tylko jeden wiersz z informacjami o liczbie zapamiętanych komentarzy pozytywnych i negatywnych. Tabela bez dodatkowej litery przechowuje niemalże wszystkie informacje potrzebne do wyliczenia wag słów w kategorii. Sama waga słowa nie jest przechowywana, by zapewnić inkrementację wartości dla poszczególnych słów. Tabela przechowuje zarówno dane dla komentarzy pozytywnych (końcówka `p`), jak i negatywnych (końcówka `n`).



Rysunek 5.2. Schemat tabel klasyfikatora DCM+

5.3.2. Uczenie wstępne

Uczenie wstępne na samym początku czyści jedną z tabel bazy danych i zeruje drugą, by wszystko rozpocząć od nowa. Następnie konwertuje otrzymane komentarze treningowe z postaci oryginalnej na wektory słów z częstościami. Po każdej konwersji od razu wywołuje metodę aktualizacji inkrementacyjnej, by dodać komentarz do bazy wiedzy.

Przekazywany wektor słów jest wcześniej konwertowany zgodnie z wzorem (4.12) z podrozdziału 4.4.5 (zmianie ulegają wartości wag).

Faza uczenia wstępnego w inkrementacyjnej wersji klasyfikatora DCM nie różni się znacząco od fazy uczenia inkrementacyjnego, z tym że od razu uczona jest cała partia dokumentów. Konkretny sposób umieszczania i aktualizacji danych został opisany w dalszej części tekstu.

5.3.3. Klasyfikacja

Metoda klasyfikująca przetwarza komentarz z wersji oryginalnej na wektor. Jeśli na tym etapie zostanie zwrócona wartość `false`, oznacza to, że komentarz zawiera wulgaryzmy, adres URL lub e-mail, albo jest pusty. Metoda dla takiego przypadku od razu przypisuje komentarz do kategorii negatywnej.

Kolejny etap to określenie oceny komentarza dla kategorii pozytywnej i negatywnej. Ponieważ w bazie danych przechowywane są tylko niektóre informacje, metoda musi przystąpić do wyliczenia wagi słów dla kategorii. Wydobywa z bazy wartość dotyczące słów, które występują w analizowanym komentarzu. Dla każdego zwróconego przez bazę słowa algorytm wylicza ostateczną wagę słowa dla kategorii. W międzyczasie sumuje wagi, by uzyskać wartości potrzebne później do wyliczenia podobieństwa. Po pobraniu wszystkich informacji z bazy danych i wyliczeniu sum określa podobieństwo komentarza do wskazanej jako parametr kategorii, używając współczynnika Jaccarda (wzór (4.17)).

Po uzyskaniu podobieństw do obu kategorii komentarzy algorytm sprawdza, czy któraś z nich jest większa od drugiej o co najmniej próg wskazany przy inicjalizacji klasy. Jeśli tak, zwraca informację o określeniu komentarza jako pozytywnego lub negatywnego. W przeciwnym razie informuje o braku rozstrzygnięcia co do rodzaju komentarza.

5.3.4. Uczenie inkrementacyjne

Uczenie najpierw modyfikuje zapamiętywany wektor słów z częstościami, by zamienić istniejące wartości zgodnie z wzorem (4.12). Po tej operacji uaktualnia informację o liczbie dokumentów należących do aktualizowanej kategorii. Następnie pobiera z bazy danych nową wartość liczby słów i wylicza mianownik wzoru na WC_{ik} (4.14).

Kolejny etap polega na aktualizacji wartości df_{ik} oraz sumy w_{id} dla słów z uczonego dokumentu. Jeśli dane słowo nie istniało wcześniej w bazie wiedzy, zostaje dodane z wykorzystaniem wartości inicjujących. Operacja dodania i aktualizacji korzysta z jednego

zapytania SQL, które w zależności od istnienia słowa dokonuje aktualizacji lub wstawienia (nie jest potrzebna operacja `SELECT` sprawdzająca istnienie słowa).

Pozostaje jeszcze uaktualnić inne dane statystyczne zawarte w bazy wiedzy, przede wszystkim informację o rozkładzie słowa w poszczególnych kategoriach (CC_i) i względnej istotności cech w kategorii (WC_{ik}). Ponieważ dodanie nowego dokumentu zmienia wspomniane wartości dla wszystkich istniejących w bazie wiedzy słów (a nie tylko tych z uczonego dokumentu), algorytm wykonuje polecenie SQL aktualizujące globalnie obie wartości. Dzięki rozbudowanym funkcjom MySQL stosunkowo złożone wyliczenia wykonuje tylko i wyłącznie baza danych. Poniżej zostało przedstawione wykorzystywane w tym celu zapytanie, gdy klasyfikator uczy się komentarza pozytywnego.

```
UPDATE `datavect_X-2` SET WCikp = LOG2(dfikp+1)/$logNk, CCi =
LOG2((2 * GREATEST(dfikp,dfikn)) / (dfikp + dfikn));
```

Tego rodzaju zoptymalizowane zapytanie pozwala uniknąć wyliczania danych w powolnym języku skryptowym PHP.

5.4. ALGORYTM K-NN

5.4.1. Modyfikacje względem oryginalnego algorytmu

Zastosowany w pracy algorytm k-NN został w kilku miejscach zmodyfikowany w porównaniu z oryginałem. Zmiany podyktowane były chęcią poprawienia jakość klasyfikacji i lepszemu dostosowaniu do rozwiązania inkrementacyjnego.

Pierwsza ze zmian dotyczy wprowadzenia algorytmu LRU w celu usuwania dokumentów, które od dłuższego czasu nie uczestniczyły w żadnej klasyfikacji. Przez uczestnictwo rozumie się wkład dokumentu treningowego w wyliczanie oceny wykorzystywanej do kategoryzacji dokumentu. Usuwanie zgodnie z zasadami LRU ma dwa cele:

- ułatwić dryft wiedzy wraz z pojawianiem się nowych komentarzy;
- ograniczyć czas potrzebny na klasyfikację, gdyż w k-NN jest on proporcjonalny do liczby pamiętanych dokumentów.

System klasyfikacji zapamiętuje do N dokumentów. Gdy ta liczba zostanie przekroczona, usuwa ten dokument, który najdłużej nie uczestniczył w klasyfikacji. Dodatkowo przy każdej klasyfikacji zerowany jest licznik dokumentów, które brały udział w klasyfikacji i zwiększany o jeden licznik tych, które w nim udziału nie brały.

Kolejna nowość względem oryginalnego algorytmu polega na zmianie miary podobieństwa. Wykorzystywana jest taka sama miara, jak w algorytmie DCM, czyli współczynnik Jaccarda.

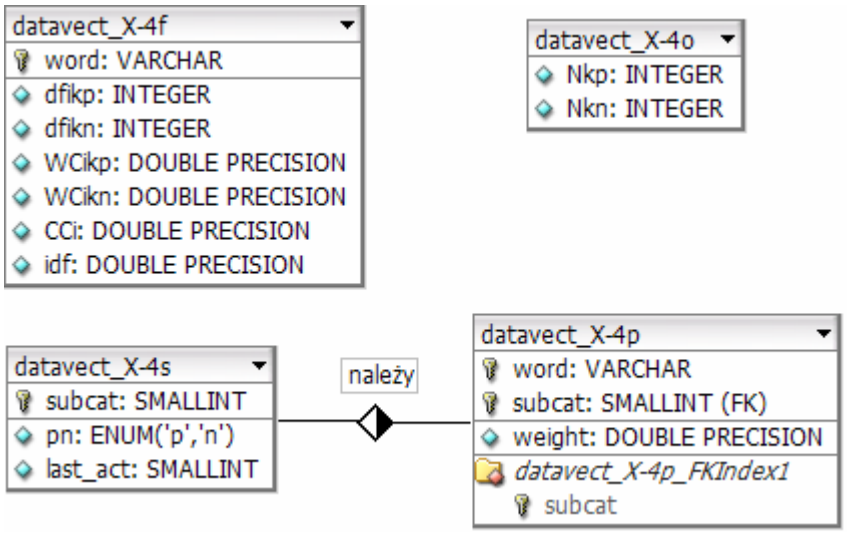
Ostatnia modyfikacja wprowadza nowy sposób liczenia wag słów w dokumencie i uwzględnia liczbę powtórzeń słowa, globalną częstość wystąpień słowa w zbiorze wiedzy oraz względną, dyskryminacyjną istotność słowa w kategorii. Wzór (5.2) przedstawia konkretny sposób wyliczania wagi słowa.

$$w_{ik} = \log(f_{ik} + 0{,}5) * \log\left(\frac{N}{n_i}\right) * 2CC_i \tag{5.2}$$

We wzorze w_{ik} to waga słowa i z kategorii k, f_{ik} to częstość wystąpienia słowa w dokumencie, N to łączna liczba dokumentów, a n_i to liczba dokumentów, w których słowo i wystąpiło co najmniej raz. CC_i jest wyliczane zgodnie z wzorem (4.15). W porównaniu z tradycyjnym systemem TFIDF wprowadzono logarytmowanie częstości wystąpień słowa, by n-krotnie występujące słowo nie miało n-krotnie większego znaczenia. Część IDF pozostała bez zmian. Część CC_i ma za zadanie zwiększać lub zmniejszać wagę słowa w zależności od tego, czy występuje ono tylko w jednej wybranej kategorii (wartość największa), czy też jest równo rozłożone w wielu kategoriach (wartość najmniejsza). Element ten został zapożyczony z algorytmu DCM.

5.4.2. Organizacja tabel bazy danych

Algorytm k-NN wykorzystuje cztery tabele, z czego dwie są tabelami pomocniczymi i przechowują informacje niezbędne do łatwego wyliczania modyfikatorów wag słów. Tabela kończąca się na `o` jest tabelą pomocniczą zawierającą zawsze tylko jeden wiersz z informacjami o liczbie zapamiętanych komentarzy pozytywnych i negatywnych (informację tę można by również uzyskać ze zliczenia wierszy tabeli `s` przy filtrowaniu pozostawiającym odpowiednią kategorię). Tabela z literą `f` również jest tabelą pomocniczą. Przechowuje wszystkie pozostałe informacje potrzebne do uzyskania modyfikatorów wag słów w kategorii, jak i same modyfikatory wyliczane ponownie po aktualizacji bazy wiedzy. Tabela z literą `s` na końcu przechowuje informację o identyfikatorze zapamiętanego dokumentu i przypisaniu tekstu do kategorii. Dodatkowo pamiętana jest liczba klasyfikacji, w których nie uczestniczył ostatnio dokument. Właściwe dane konkretnego dokumentu (wektor słów z wagami) zawiera tabela kończąca się literą `p`. Rysunek 5.3 przedstawia wszystkie tabele.



Rysunek 5.3. Schemat tabel klasyfikatora k-NN

5.4.3. Uczenie wstępne

Uczenie wstępne rozpoczyna się od wyczyszczenia trzech tabel bazy danych (tabel z informacjami o dokumentach i tabeli pomocniczej wag słów) i wyzerowania czwartej. Następnie konwertuje się otrzymane komentarze treningowe z postaci oryginalnej na wektory słów z częstościami i jednocześnie tworzy listę słów występujących we wszystkich dokumentach pozytywnych lub negatywnych wraz z liczbą dokumentów, w których się pojawiły.

Kolejny krok to umieszczenie w tabeli pomocniczej listy słów wraz z informacją o ich wystąpieniach w dokumentach (df_{ik}). Dzięki tej operacji wystarczy wykonać jedno aktualizacyjne zapytanie SQL, by wyliczyć modyfikatory wag. Wyliczone modyfikatory dla poszczególnych słów pobiera się z bazy danych, by dla każdego dokumentu wyliczyć wagi dla zawartych w nim słów.

Ostatni etap to zapis poszczególnych dokumentów treningowych jako wektorów cech. Zapis wykonywany jest najpierw do tabeli informacyjnej dokumentu (końcówka `s`). Po uzyskaniu unikatowego identyfikatora dokumentu poszczególne słowa z wagami zostają wpisane do tabeli z końcówką `p`.

5.4.4. Klasyfikacja

Metoda klasyfikująca przyjmuje komentarz w wersji oryginalnej i przetwarza go na wektor. Jeśli na tym etapie zostanie zwrócona wartość `false`, komentarz zapewne zawie-

ra wulgaryzmy, adres URL lub e-mail, albo jest pusty. W takiej sytuacji metoda od razu kończy działanie i uznaje komentarz za negatywny.

Kolejny etap to określenie oceny komentarza dla kategorii pozytywnej i negatywnej. W algorytmie k-NN dokumenty z fazy uczenia są pamiętane jako wektory cech z wagami. Nie trzeba przeprowadzać żadnych dodatkowych wyliczeń na tym etapie. Kod pobiera z bazy danych identyfikator i typ (pozytywny/negatywny) dokumentów treningowych. Następnie na podstawie tych identyfikatorów pobiera wektor słów reprezentujący dokument i wylicza jego podobieństwo do klasyfikowanego dokumentu. Uzyskane wartości podobieństwa wraz z identyfikatorem zostają zapamiętane.

Po wyliczeniu wszystkich podobieństw algorytm sortuje je malejąco i pozostawia tylko k pierwszych z nich. Następnie osobno sumuje wartości podobieństwa dotyczące wzorcowych komentarzy pozytywnych i negatywnych. Na końcu wartości te zwraca do kodu wywołującego metodę.

Ostatni fragmenty kodu metody inkrementuj licznik związany z LRU dla tych komentarzy wzorcowych, które nie brały udziału w sumowaniu podobieństw. Licznik pozostałych komentarzy ustawia na 0.

Algorytm testuje zebrane sumy podobieństw do obu kategorii, by poznać, czy któraś z nich jest większa od drugiej o więcej niż próg wskazany przy inicjalizacji klasy. Jeśli tak, zwraca informację o określeniu komentarza jako pozytywnego lub negatywnego. W przeciwnym razie informuje o braku rozstrzygnięcia co do rodzaju komentarza.

5.4.5. Uczenie inkrementacyjne

Uczenie najpierw uaktualnia informację o liczbie dokumentów należących do aktualizowanej kategorii. Następnie stosując odpowiednie zapytania SQL, aktualizuje lub wprowadza wartości df_{ik}, WC_{ik}, CC_i oraz IDF, by obejmowały nowy dokument uczący. Wyliczone na tym etapie wartości CC_i oraz IDF są pobierane przez kod z bazy danych, by mogły być użyte do określenia wag słów dodawanego dokumentu.

Na podstawie zebranych danych statystycznych algorytm wylicza wago słów a następne je normalizuje. W kolejnym kroku dodaje nowy komentarz uczący do bazy danych, umieszczając najpierw w tabeli z końcówką `s` informację o jego rodzaju a następnie wprowadzając do bazy danych poszczególne słowa i ich znaczenia (tabela z końcówką `f`).

Jeśli liczba przechowywanych dokumentów jest większa od zadanej przez operatora wartości, kod przystępuje do usunięcia komentarza uczącego, który najdawniej nie uczest-

niczył w klasyfikacji. Znajduje dokument z największą wartością licznika, pobiera jego identyfikator a następnie go usuwa.

5.5. ALGORYTM ICNN — NOWA PROPOZYCJA

5.5.1. Nowy algorytm na tle swego pierwowzoru

Nowa propozycja autora pracy dotycząca algorytmu klasyfikacji tekstu wzoruje się na pomyśle przedstawionym w pracy [14] (kNN) i opisanym w podrozdziale 4.6.3, który stanowi rozwinięcie algorytmu GIS starającego się połączyć zalety klasyfikatorów Rocchio i k-NN. Oryginalny algorytm kNN nie zapewnia inkrementacyjnego sposobu uczenia. Redukuje dokumenty do kilkunastu globalnych grup uogólnionych na etapie uczenia i nie informuje o sposobie dołączania do nich nowych dokumentów lub też późniejszego, dynamicznego redukowania grup. W niniejszej pracy opracowano algorytm inkrementacyjnego tworzenia i łączenia grup oraz ich usuwania z wykorzystaniem mechanizmu LRU. Dodatkowo zmieniono sposób miary podobieństwa i określania wag słów w taki sam sposób, jak w implementowanym algorytmie k-NN (patrz podrozdział 5.4.1).

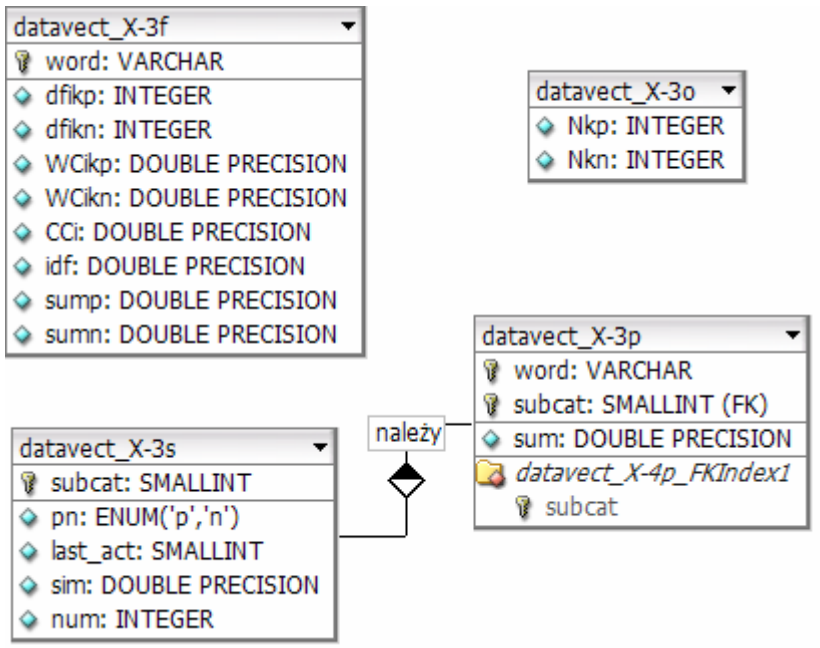
Algorytm uczenia wstępnego algorytmu ICNN został niemalże w całości zaimplementowany na podstawie algorytmu konstrukcji modelu przedstawionego w [14]. Zmiany dotyczyły jedynie tworzenia pełnej globalnej wersji kategorii na podstawie wszystkich dotyczących jej dokumentów uczących. W oryginalnym algorytmie była ona tworzona w taki sam sposób jak grupy uogólnione, ale z maksymalnym progiem podobieństwa. Tutaj globalna wersja kategorii powstaje niemalże automatycznie, gdyż wymaga tego liczenie wartości CC_i i IDF. Na etapie klasyfikacji jest ona traktowana w ten sam sposób, co w oryginale, ale nie może zostać usunięta. Globalna wersja jest w znacznym stopniu podobna do tej z algorytmu DCM.

Całkowicie nowym elementem ICNN w porównaniu z pierwowzorem jest możliwość inkrementacyjnej aktualizacji bazy wiedzy. Operacja ta przeprowadzana jest w kilku krokach, które zostały szczegółowo opisane w podrozdziale 5.5.5.

5.5.2. Organizacja tabel bazy danych

Algorytm ICNN stosuje cztery tabele, z czego jedna jest tabelą pomocniczą przechowującą informacje o liczbie komentarzy pozytywnych i negatywnych uwzględnianych w globalnym wektorze wag. Jest to tabela kończąca się na `o`. Tabela z literą `f` jest zarówno

tabelą pomocniczą przy określaniu docelowych wag słów, jak i stanowi wektor globalny cech (osobny dla dokumentów poprawnych i niepoprawnych). Tabela z literą `s` na końcu przechowuje informację o identyfikatorze uogólnionej grupy dokumentów, przypisaniu jej do kategorii, obszarze obejmowania (minimalnym podobieństwie) oraz liczniku nieaktywności w klasyfikacji. Właściwe dane uogólnione grupy (wektor słów z wagami) zawiera tabela kończąca się literą `p`. Rysunek 5.4 przedstawia wszystkie tabele.



Rysunek 5.4. Schemat tabel klasyfikatora ICNN

5.5.3. Uczenie wstępne

Kod zaczyna uczenie wstępne od usunięcia danych z trzech tabel bazy danych (tabel z informacjami o dokumentach i tabeli pomocniczej wag słów) i wyzerowania czwartej tabeli. Następnie konwertuje otrzymane komentarze treningowe z postaci oryginalnej na wektory słów z częstościami i jednocześnie tworzy listę słów występujących we wszystkich dokumentach pozytywnych lub negatywnych wraz z liczbą dokumentów, w których się pojawiły.

Kolejny krok to umieszczenie w tabeli pomocniczej listy słów wraz z informacją o ich wystąpieniach w dokumentach (df_{ik}). Dzięki tej operacji wystarczy wykonać jedno aktualizacyjne zapytanie SQL, by wyliczyć modyfikatory wag. Wyliczone modyfikatory

dla słów służą następnie do wyliczenia docelowych wag słów poszczególnych dokumentów.

Kolejny etap to znajdowanie lokalnego sąsiedztwa dla każdego komentarza. Ten krok i kolejne wykonuje się osobno dla komentarzy poprawnych i niepoprawnych. Dla każdego dokumentu zostaje wyliczone podobieństwo do innych dokumentów treningowych. Jeśli podobieństwo obu dokumentów jest większe od zadanego minimum, zostają one zapamiętane jako lokalna grupa sąsiedzka. W ten sposób powstaje macierz podobieństw.

Następnie wyszukany zostaje ten dokument, które ma najliczniejsze sąsiedztwo. Na podstawie jego oraz sąsiedztwa utworzony zostaje nowy uogólniony wektor cech łączący w sobie dokumenty z danej grupy lokalnej. Do wykonania uogólnienia wykorzystywany jest algorytm centroidalny (patrz wzór (4.11)). Wektor jest porównywany z oryginalnymi dokumentami, na podstawie których powstał, by poznać zasięg oddziaływania całej grupy. Po wykonaniu tych czynności uogólniony wektor cech zostaje zapisany najpierw do tabeli informacyjnej dokumentu (końcówka `s`). Po uzyskaniu unikatowego identyfikatora dokumentu poszczególne słowa z wagami zostają wpisane do tabeli z końcówką `p`. Po tej operacji z macierzy podobieństwa zostają usunięte te dokumenty, na podstawie których została utworzona grupa.

Cała operacja opisana w poprzednim akapicie powtarza się do momentu, w którym macierz będzie pusta lub też gdy pozostaną w niej jedynie dokumenty nie mające w lokalnym sąsiedztwie nikogo innego. Pojedyncze dokumenty zostają zapamiętane jako osobne grupy z zakresem oddziaływania równym minimum podobieństwa wykorzystywanym w trakcie określania lokalnych grup.

5.5.4. Klasyfikacja

Metoda klasyfikująca pobiera komentarz w wersji podanej przez użytkownika i przetwarza go na wektor. Jeśli na tym etapie zostanie uzyskana wartość `false`, komentarz prawdopodobnie zawiera wulgaryzmy, adres URL, adres e-mail, lub też jest pusty. W takiej sytuacji metoda od razu kończy działanie i uznaje komentarz za negatywny.

Kolejny etap to określenie oceny komentarza dla kategorii pozytywnej i negatywnej. Najpierw wyliczone zostaje podobieństwo komentarza do globalnych reprezentantów (wektorów) wpisów poprawnych i niepoprawnych. Wyliczenie podobieństwa z użyciem współczynnika Jaccarda wymaga wcześniejszego przeprowadzenia kilku obliczeń, gdyż

wagi słów wektorów globalnych nie są pamiętane w sposób bezpośredni (by zapewnić łatwość aktualizacji). Uzyskane podobieństwa stanowią początkowe wartości oceny komentarza (tworzone są dwie oceny, pozytywna i negatywna).

Po tej fazie kod przystępuje do sprawdzania podobieństwa analizowanego komentarza do grup uogólnionych. Pobiera z bazy danych identyfikator i typ (pozytywny/negatywny) grupy. Następnie na jego podstawie pobiera wektor słów reprezentujący grupę i wylicza podobieństwo wektora do klasyfikowanego dokumentu. Uzyskana wartość podobieństwa jest porównywana z minimalnym podobieństwem dotyczącym grupy. Jeśli jest większa od minimum, oznacza to, że klasyfikowany dokument znalazła się w obrębie oddziaływania grupy. W takiej sytuacji podobieństwo zostaje pomnożone przez zlogarytmowaną liczbę dokumentów tworzących grupę (w ten sposób większe znaczenie i wpływ na wynik ma podobieństwo do uogólnionej grupy zrzeszającej więcej dokumentów, ale wpływ ten nie rośnie liniowo). Uzyskana wartość jest dodawana do wynikowej oceny komentarza (jeśli grupa była pozytywna, to do pozytywnego; jeśli była negatywna, do negatywnego).

Przedostatni fragmenty kodu inkrementuje licznik związany z LRU dla tych grup uogólnionych, które nie brały udziału w sumowaniu podobieństw. Licznik pozostałych grup ustawia na 0.

Algorytm testuje oceny zebrane dla obu kategorii, by poznać, czy któraś z nich jest większa od drugiej o więcej niż próg wskazany przy inicjalizacji klasy. Jeśli tak, zwraca informację o określeniu komentarza jako pozytywnego lub negatywnego. W przeciwnym razie informuje, iż nie potrafił rozstrzygnąć, czy komentarz jest poprawny, czy też niepoprawny.

5.5.5. Uczenie inkrementacyjne

Oryginalny algorytm kNN, na którym wzoruje się klasyfikator ICNN nie dopuszczał uczenia inkrementacyjnego. Podany przepis aktualizacji zbioru danych został opracowany na potrzeby niniejszej pracy przez jej autora. Kolejne kroki algorytmu zostały opisane poniżej.

1. Kod aktualizuje wykorzystywane globalne modyfikatory wag i wylicza docelowe wagi słów dla nowego komentarza uczącego. Uaktualnia również globalny wektor wag dla kategorii pozytywnej lub negatywnej (w zależności od rodzaju dodawanego komentarza).

2. Mierzy podobieństwo nowego dokumentu do poszczególnych istniejących grup pozytywnych (jeśli nowy dokument jest pozytywny) lub negatywnych (w przeciwnym razie). Etap ten jest bardzo podobny do zwykłej klasyfikacji, ale obejmuje wyszukiwanie tylko w grupach konkretnej kategorii. Kod zapamiętuje identyfikatory tylko tych grup, dla których podobieństwo było tak duże, że nowy dokument znajduje się w zasięgu ich obejmowania.
3. Jeśli nie została znaleziona żadna grupa dostatecznie podobna do komentarza, nowy dokument tworzy całkowicie nową grupę a kod stosuje dla niej obszar obejmowania równy minimalnemu podobieństwu ustalonemu przez operatora. Gdy została przekroczona maksymalna założona liczba grup, usunięta zostaje ta ze starych grup, która najdłużej nie brała udziału w żadnej klasyfikacji.
4. Jeśli zostało wykryte podobieństwo tylko do jednej grupy lub też do kilku grup, ale dwie o największym podobieństwie nie są dostatecznie blisko siebie, wtedy nowy dokument zostaje złączony z grupą o największym podobieństwie. Złączenie wykonywane jest na podstawie algorytmu centroidalnego. Określony zostaje nowy obszar zasięgu grupy — najmniejsza z wartości: starego zasięgu, nowego wyliczonego podobieństwa i minimalnego podobieństwa określonego przez operatora.
5. Jeśli co najmniej dwie grupy są dostatecznie blisko siebie i nowego dokumentu, dochodzi do scalenia grup (w podobny sposób, jak w punkcie 4., ale w scaleniu udział bierze większa liczba wektorów cech) i dołączenia do nich nowego dokumentu. Dodatkowo zostaje wyliczony nowy obszar obejmowania o minimalnym zasięgu. Nowa grupa zostaje dodana do odpowiednich tabel. Stare grupy biorące udział w scaleniu są usuwane.

6. WYNIKI

Niniejszy rozdział zawiera wyniki klasyfikacji z wykorzystaniem czterech algorytmów zaimplementowanych na potrzeby niniejszej pracy. Do ich uzyskania wykorzystano wszystkie trzy grupy komentarzy opisane w podrozdziale 2.3.

6.1. SPOSÓB PRZEPROWADZENIA BADAŃ KLASYFIKATORÓW

6.1.1. Wykorzystywane miary jakości klasyfikacji

Ponieważ w algorytmach klasyfikacji wprowadzono możliwość odmówienia skategoryzowania komentarza, jeśli jego oceny dla kategorii pozytywnej i negatywnej są podobne, należy zastanowić się na właściwym doborem miary klasyfikacji. Warto również zastanowić się, która informacja byłaby najważniejsza w przypadku klasyfikacji komentarzy.

Wydaje się, iż z punktu widzenia moderatora forum stosującego przedstawiane klasyfikatory najważniejszy byłby błąd dotyczący sytuacji, w której niepoprawny komentarz przedostałby się i został wyświetlony na forum. Innymi słowy, moderatorowi zależy przede wszystkim na minimalizacji wartości FP (ang. *false positive*), patrz tabela 4.1. Z punktu widzenia dobrze zachowującego się użytkownika forum ważnym jest, by nie został odrzucony jego poprawny komentarz. Częste odrzucanie komentarzy poprawnych może go mocno zniechęcić. Zadowolenie użytkowników z witryny jest istotne dla moderatora, więc i on powinien być zainteresowany minimalnym odrzucaniem komentarzy poprawnych, choć nie wpływają one bezpośrednio na nakład jego pracy. Drugim elementem jest więc minimalizacja wartości FN (ang. *false negative*), patrz tabela 4.1.

Z punktu widzenia moderatora największy nakład pracy będzie dotyczył komentarzy, co do których klasyfikator odmówił kategoryzacji. Nadajmy tego rodzaju komentarzom skrót UC (ang. *unclassified*). Moderator chciałby z jednej strony zminimalizować wartość UC a z drugiej wartości FP i FN. Najczęściej jednak nie jest to możliwe, gdyż są one ze sobą mocno powiązane (wzrost jednej zmniejsza pozostałe).

Z przedstawionego opisu wynika, iż należy skupić się nie na wartościach klasyfikacji poprawnej, ale na uzyskiwanych błędach i odmowach klasyfikacji. Sposób procentowego liczenia błędów oraz odmówień przedstawiają wzory (6.1), (6.2) i (6.3).

$$error_{\mathrm{N}} = \frac{FP}{TP + FP + TN + FN + UC} * 100\% \tag{6.1}$$

$$error_{\mathrm{P}} = \frac{FN}{TP + FP + TN + FN + UC} * 100\% \tag{6.2}$$

$$uncla = \frac{UC}{TP + FP + TN + FN + UC} * 100\% \tag{6.3}$$

Aby zapewnić porównanie zaimplementowanych algorytmów z wartościami uzyskiwanymi przez inne systemy klasyfikujące, wyliczona zostanie również wartość F_1 w wersji makrouśrednionej z wykorzystaniem wzoru (6.4). Wartość ta będzie liczona tylko dla testów, w których wyłączone zostanie odmawianie kategoryzacji przez klasyfikatory.

$$F_{1M} = \left(\frac{2 * precision_P * recall_P}{precision_P + recall_P} + \frac{2 * precision_N * recall_N}{precision_N + recall_N} \right) * \frac{1}{2} \tag{6.4}$$

6.1.2. Sposób przeprowadzenia testów

Kod testujący został napisany w taki sposób, by wszystkie cztery klasyfikatory w danym przebiegu testującym otrzymywały komentarze w dokładnie tej samej kolejności. Jest to bardzo ważne, gdyż w przypadku błędów lub odmówienia przypisania algorytm uczy się, co wpływa na jego przyszłe klasyfikacje. Stosowanie tych samych komentarzy zapewnia porównywanie w uczciwy sposób. Pozwala też sprawdzić, czy wszystkie dobrze radzą sobie przy jednej kolejności a gorzej przy innej, czy może każdy klasyfikator lepiej sprawdza się w jednej szczególnej kolejności nadsyłania komentarzy.

Większość algorytmów klasyfikacji korzysta z dodatkowych parametrów sterujących ich pracą. Przed przeprowadzeniem właściwych testów postarano się tak dobrać te parametry, by uzyskać jak najlepsze rezultaty dla konkretnych zbiorów komentarzy. Wartości te przedstawia tabela 6.1.

Kod testujący do fazy inicjacyjnej (wstępne uczenie) pobiera z bazy danych ustaloną przez testującego liczbę komentarzy pozytywnych i negatywnych. Pozostałe komentarze są wykorzystywane w fazie testowania z uczeniem inkrementacyjnym. Ich dobór jest losowy, co przy kilkukrotnych testach pozwala dowiedzieć się, w jaki sposób kolejność nadchodzenia komentarzy zmienia jakość klasyfikatora. Im odchyłka między najlepszym i najgorszym rezultatem klasyfikatora mniejsza, tym lepiej.

Tabela 6.1. Parametry algorytmów stosowane w trakcie testów

Algorytm	Komentarze gry.pl	Komentarze gsmonline.pl	Komentarze kucharskie
Winnow	alfa: 2 beta: 0,5 nieistotność niska: 0,005 nieistotność wysoka: 2 próg strefy niepew.: 0,5	alfa: 2 beta: 0,5 nieistotność niska: 0,005 nieistotność wysoka: 2 próg strefy niepew.: 0,5	alfa: 2 beta: 0,5 nieistotność niska: 0,005 nieistotność wysoka: 2 próg strefy niepew.: 0,5
DCM+	próg strefy niepew.: 0,05	próg strefy niepew.: 0,02	próg strefy niepew.: 0,1
k-NN	próg strefy niepew.: 0,5 k: 10 maks. komentarzy: 200	próg strefy niepew.: 0,2 k: 10 maks. komentarzy: 200	próg strefy niepew.: 0,7 k: 10 maks. komentarzy: 200
ICNN	próg strefy niepew.: 0,5 podobieństwo: 0,05	próg strefy niepew.: 0,2 podobieństwo: 0,05	próg strefy niepew.: 0,8 podobieństwo: 0,05

Faza klasyfikacji połączona jest z fazą inkrementacyjnego uczenia w bardzo prosty sposób, który nie oddaje w pełni rzeczywistych warunków, w jakich przyszłoby pracować klasyfikatorom. Po każdej klasyfikacji kod sprawdza, czy była ona poprawna. Jeśli nie, uczy się danego komentarza w sposób inkrementacyjny. Gdy klasyfikator odmówił kategoryzacji, również przystępuje do uczenia komentarza. W rzeczywistym systemie uczenie następowałoby tylko co jakiś czas w sposób zbiorczy — moderator przeglądałby komentarze niesklasyfikowane i błędnie oznaczone oraz określałby ich przynależność do kategorii. Zastosowany algorytm testujący stanowi jednak w miarę dobrą aproksymację rzeczywistości dla celów porównawczych.

Kod testujący poza zbieraniem wyników i ich zapamiętywaniem przeprowadza również pomiary szybkości działania poszczególnych etapów klasyfikacji: przygotowanie komentarzy, uczenie wstępne, klasyfikacja i uczenie inkrementacyjne.

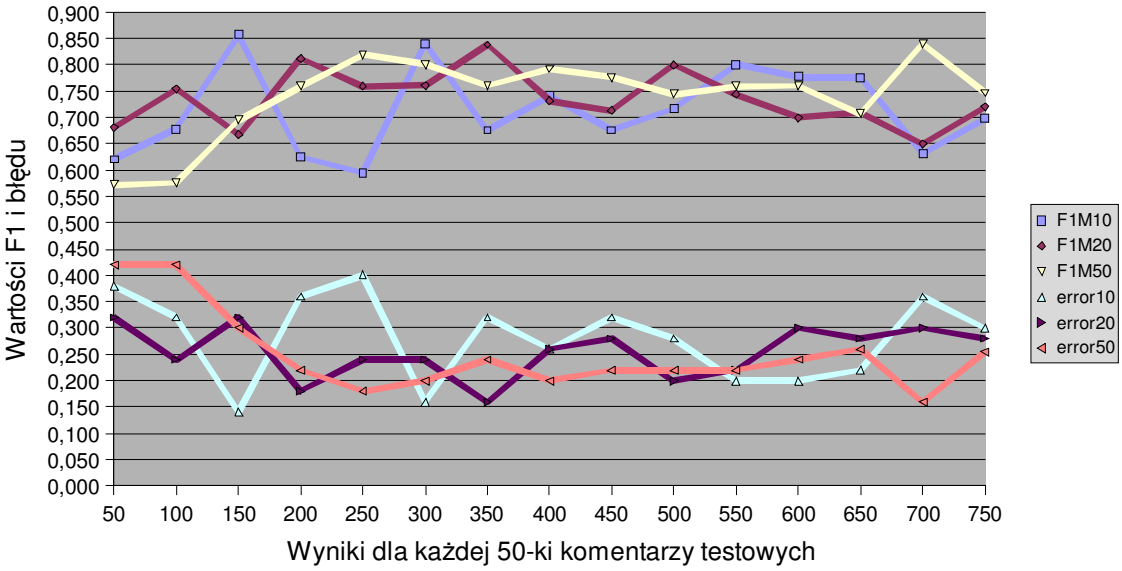
6.2. WYNIKI JAKOŚCIOWE I CZASOWE ALGORYTMÓW KLASYFIKACJI

6.2.1. Wpływ uczenia inkrementacyjnego oraz skali uczenia wstępnego

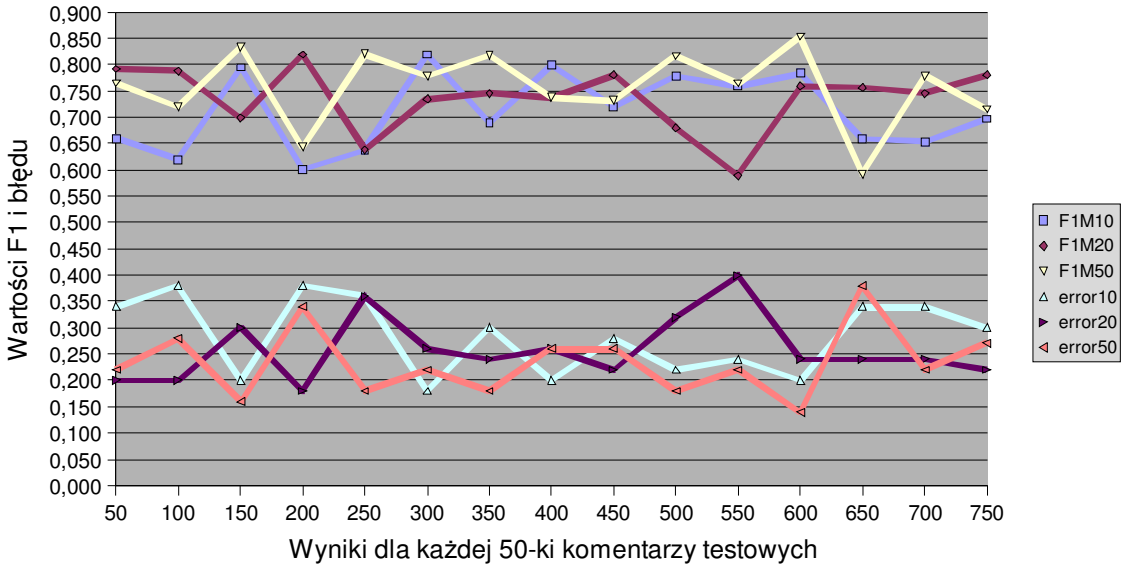
Najpierw przedstawione zostaną wykresy obrazujące, w jaki sposób na wartość uzyskiwanego błędu i miary F_1 wpływa liczba początkowych komentarzy uczących i proces uczenia inkrementacyjnego związany z nadchodzeniem nowych komentarzy. Z racji ogromnej liczby danych przedstawianych na wykresach, zostały one podzielone ze względ-

du na algorytm i typ komentarzy. Badanie przeprowadzono osobno dla włączonego i wyłączonego progu niepewności (odmawiania klasyfikacji). Ponieważ łącznie powstały 24 wykresy (2 tryby * 4 algorytmy * 3 typy komentarzy), wybrano tylko 8 najbardziej reprezentatywnych (po jednym na tryb i algorytm).

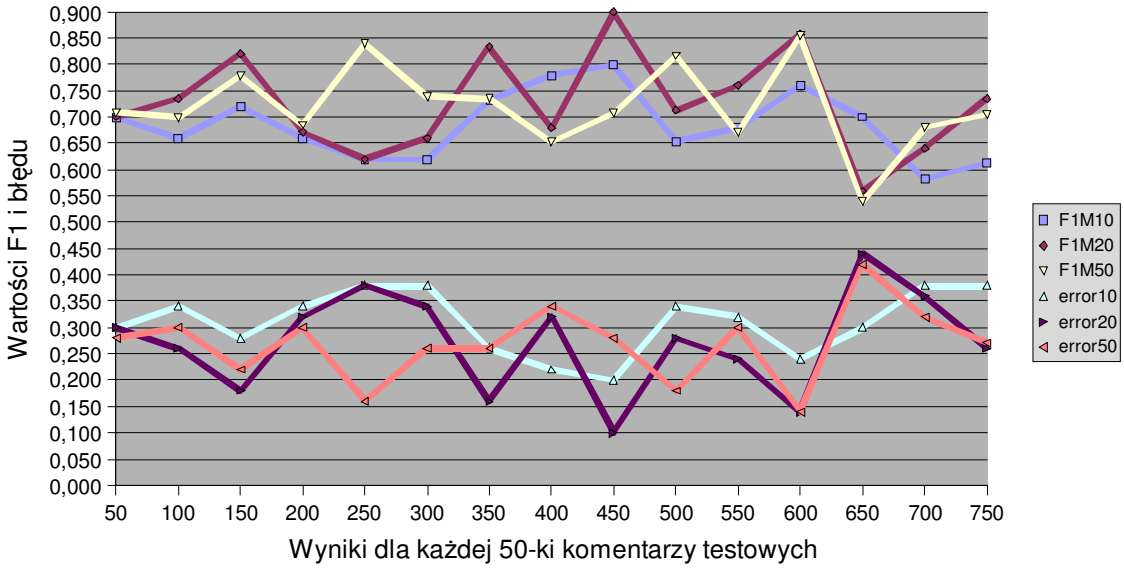
Rysunki od 6.1 do 6.4 przedstawiają uzyskane wyniki po wyłączeniu progu niepewności, czyli każdorazowej próbie wykonania klasyfikacji, nawet jeśli jej wynik mógłby być bardzo niepewny. Wyniki dotyczą najlepszego z 5 pięciu testów wykonanych dla każdego algorytmu. Wartości 10, 20 i 50 oznaczają liczbę komentarzy uczenia wstępnego (np. 10 pozytywnych i 10 negatywnych).



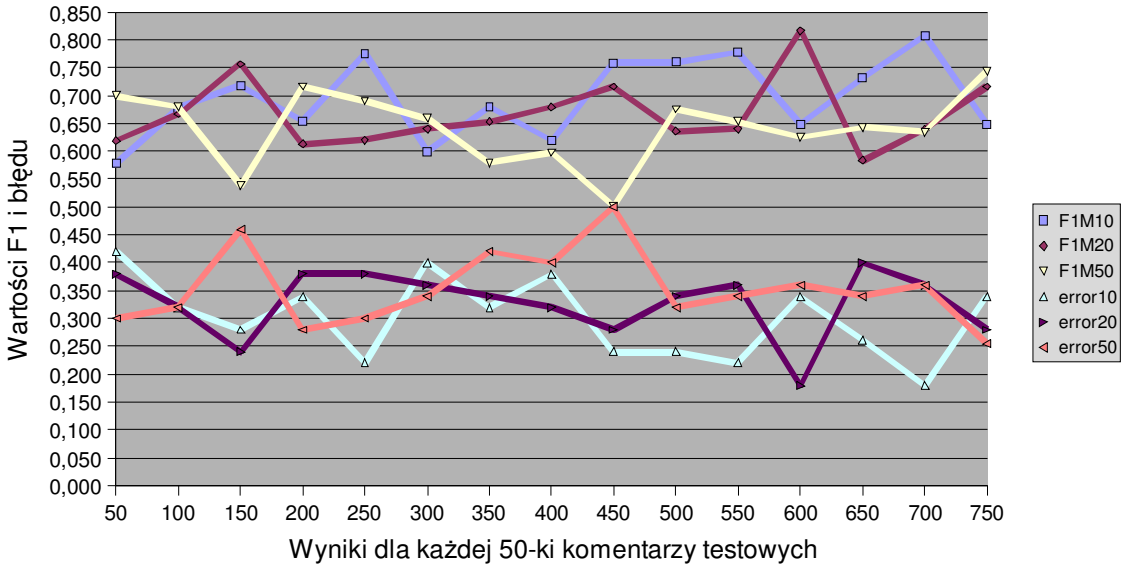
Rysunek 6.1. Wartości $F_{1\text{makro}}$ i błędu dla algorytmu Winnow i komentarzy gry.pl przy wyłączonym progu niepewności



Rysunek 6.2. Wartości $F_{1\text{makro}}$ i błędu dla algorytmu DCM+ i komentarzy gry.pl przy wyłączonym progu niepewności



Rysunek 6.3. Wartości $F_{1\text{makro}}$ i błędu dla algorytmu k-NN i komentarzy gry.pl przy wyłączonym progu niepewności

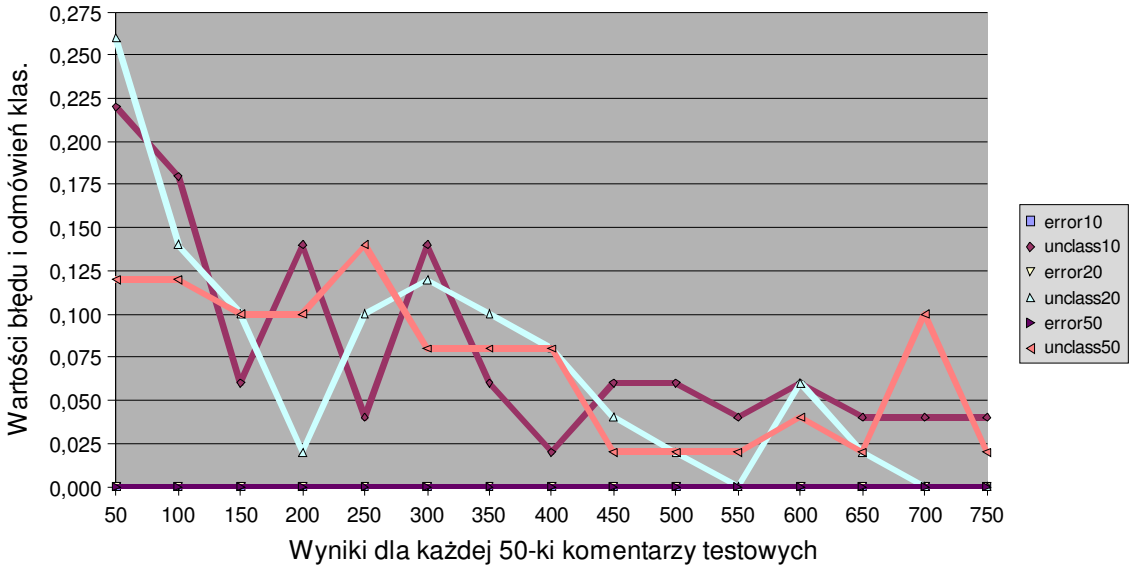


Rysunek 6.4. Wartości $F_{1\text{makro}}$ i błędu dla algorytmu ICNN i komentarzy gry.pl przy wyłączonym progu niepewności

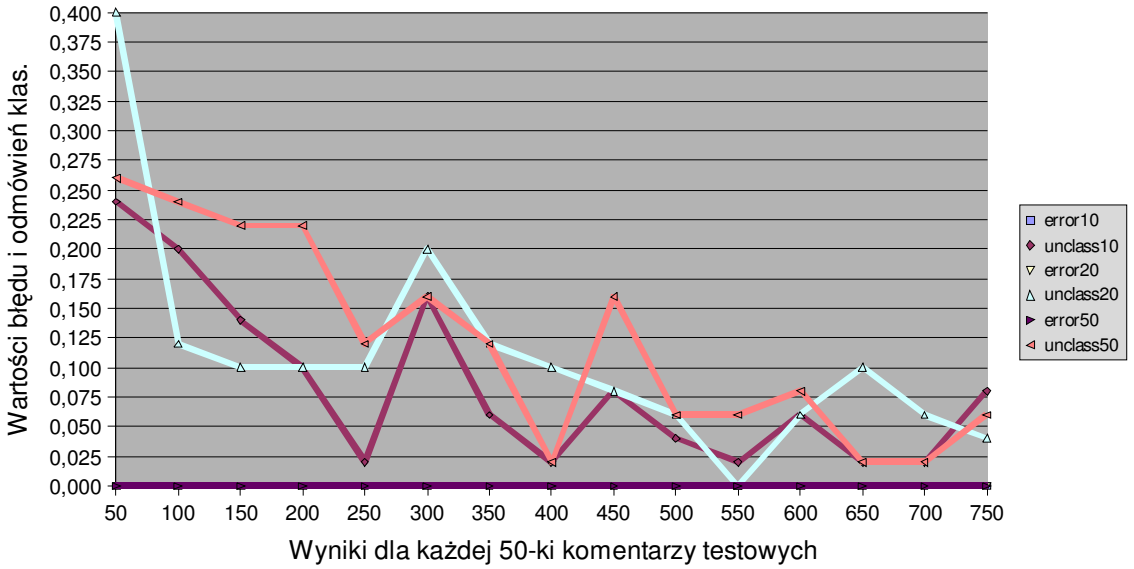
Wyniki przedstawione na 4 wykresach są zaskakujące. Żaden z czterech algorytmów po uczeniu inkrementacyjnym w znaczący sposób nie zmniejszył błędu. Można to tłumaczyć faktem występowania coraz to nowych wyrazów w kolejnych komentarzach i stosunkowo niewielką liczbą dokumentów biorących udział w testach. Testy prowadzone w innych pracach ([32]) wykazały, że uczenie inkrementacyjne rzeczywiście zmniejsza błąd, choć badanie prowadzone było na większej liczbie dokumentów. Jedyny algorytm, który wyrazie skorzystał z inkrementacyjnego uczenia, to Balanced Winnow. Nie powinno to dziwić — został zaprojektowany do właśnie takiego działania i sposobu uczenia.

Zwiększenie liczby komentarzy uczenia wstępnego w większości przypadków powodowało zmniejszenie błędu uzyskiwanego w pierwszych 50 komentarzach testowych. Z drugiej strony pojawiły się wyjątki — czasem to 20 komentarzy wstępnych uzyskiwało lepszy wynik niż 50.

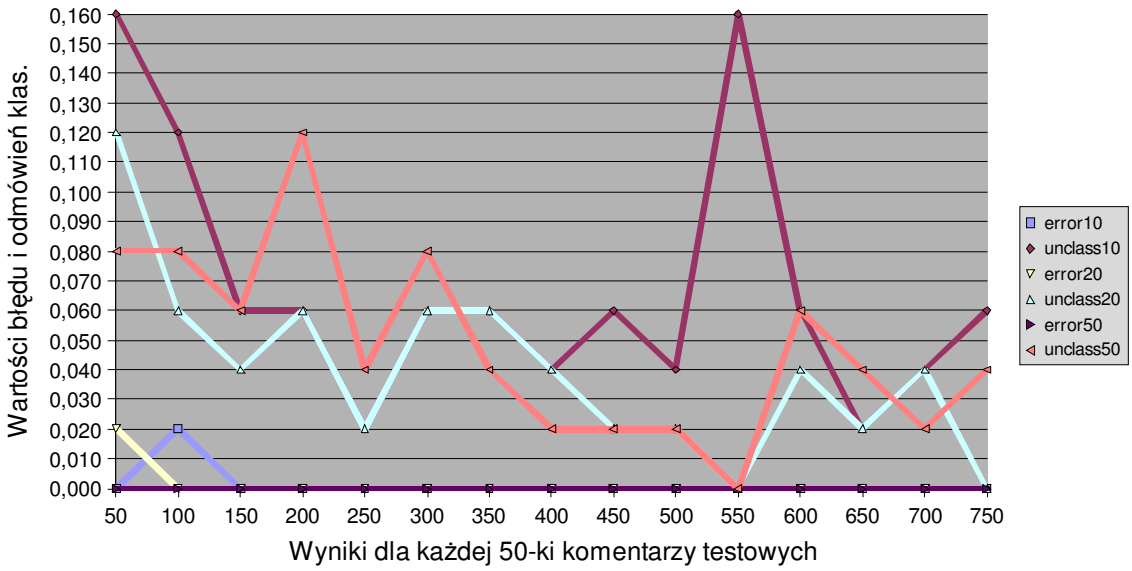
Rysunki od 6.5 do 6.8 przedstawiają wyniki uzyskane z włączonym progiem niepewności, czyli odrzucaniem komentarzy, co do których algorytm nie potrafił wedle swej wiedzy jednoznacznie przypisać kategorii. Wyniki dotyczą najlepszego z 5 pięciu testów wykonanych dla każdego algorytmu. Wartości 10, 20 i 50 oznaczają liczbę komentarzy uczenia wstępnego. Zamiast wartości F_1 przedstawiona została skala odmówień klasyfikacji (*unclass*).



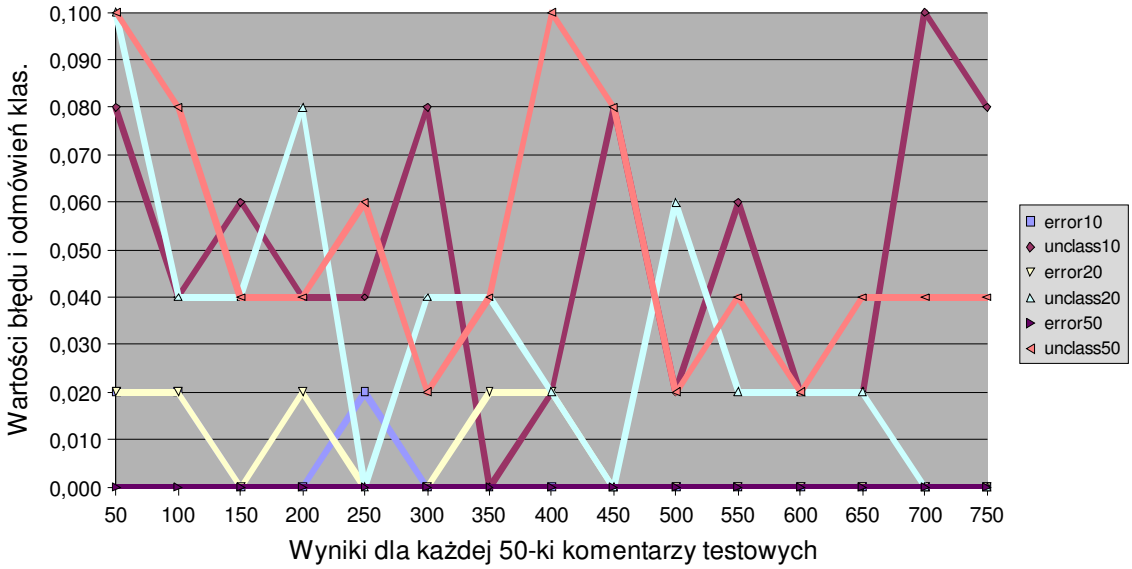
Rysunek 6.5. Wartości odmówień i błędu algorytmu Winnow dla przepisów kucharskich



Rysunek 6.6. Wartości odmówień i błędu algorytmu DCM+ dla przepisów kucharskich



Rysunek 6.7. Wartości odmówień i błędu algorytmu k-NN dla przepisów kucharskich



Rysunek 6.8. Wartości odmówień i błędu algorytmu ICNN dla przepisów kucharskich

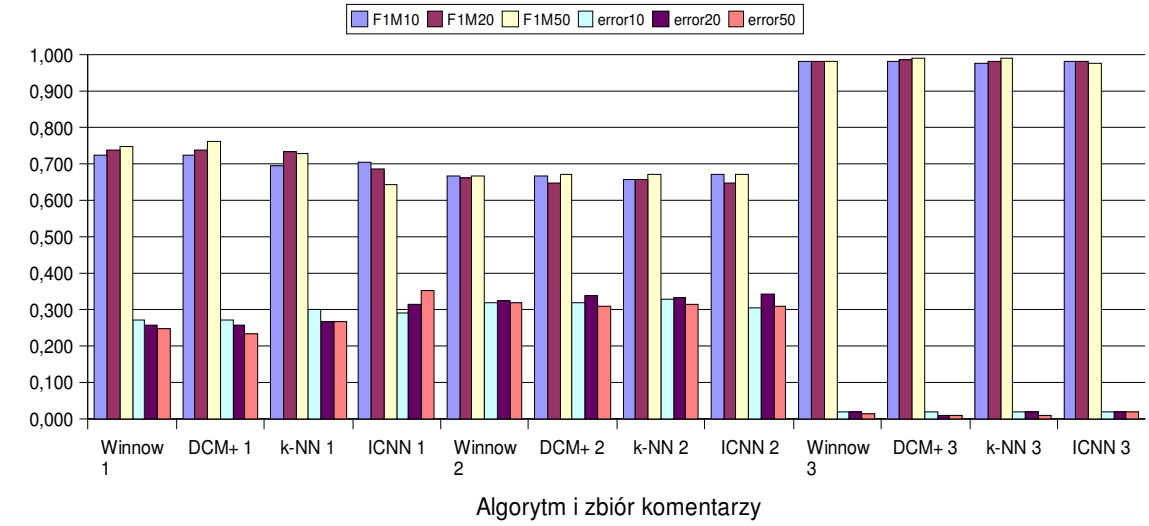
Algorytmy Winnow i DCM+ wraz ze zwiększaniem bazy wiedzy dla przedstawionego przykładu z przepisami kucharskimi zmniejszały liczbę komentarzy traktowanych jako niepewne. Proces ten jest łatwo zauważalny. W algorytmie k-NN również jest on widoczny, choć w mniejszym stopniu. W algorytmie ICNN trudno się doszukać zmniejszania liczby odmówień. Co więcej, działa on wyjątkowo nierównomiernie.

W algorytmach Winnow i k-NN zwiększenie liczby komentarzy uczenia wstępnego zaowocowało mniejszą niepewnością przy pierwszych 50 komentarzach. Ciekawe jest działanie algorytmu DCM+, w którym zwiększenie komentarzy zwiększyło również nie-

pewność. Może to oznaczać, iż wyjątkowo szybko przyswaja dokumenty uczące, co później sprawia mu problemy, gdy pojawiają się w klasyfikowanych dokumentach nowe wyrazy. Algorytm ICNN zachowywał się podobnie niezależnie od liczby komentarzy uczenia wstępnego.

6.2.2. Wpływ uczenia wstępnego i porównanie najlepszych wyników klasyfikatorów

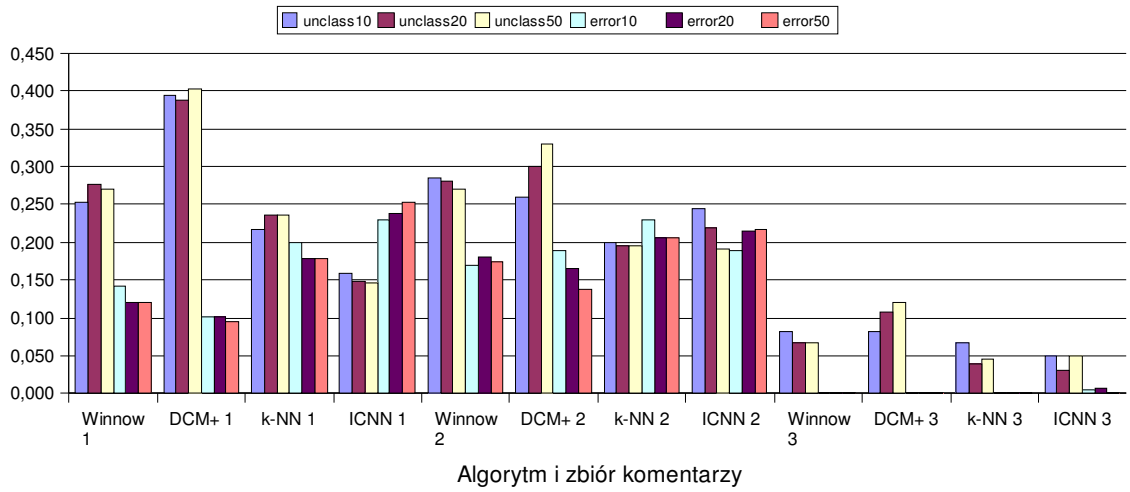
Kolejny zestaw wykresów dotyczyć będzie zobrazowania wpływu uczenia wstępnego na jakość klasyfikacji. Dodatkowo zostaną przedstawione najlepsze wyniki (w pięciu przeprowadzonych testach) uzyskane przez poszczególne klasyfikatory dla wszystkich zbiorów komentarzy. Rysunek 6.9 przedstawia wykres dotyczący testów z wyłączonym odmawianiem kategoryzacji.



Rysunek 6.9. Najlepsze wyniki klasyfikatorów w przeprowadzonych testach z wyłączonym progiem niepewności (numeracja: 1 – gry.pl, 2 – gsmonline.pl, 3 – kucharskie)

Warto zwrócić uwagę, że większość algorytmów klasyfikacji poprawiała swoje wyniki, gdy zastosowano uczenie wstępne zwiększą liczbą wyników. Wyjątkiem okazał się algorytm ICNN, w którym większy początkowy zbiór uczący w ogóle nie poprawiał wyników a czasem je nawet pogarszał. Algorytm Winnow także słabo reagował na większe wartości zbioru, ale w odróżnieniu od ICNN jego zwiększenie nie wiązało się z pogorszeniem jakości klasyfikacji.

Rysunek 6.10 przedstawia wykres dotyczący testów z włączonym odmawianiem. Podobnie jak w poprzednim podrozdziale zamiast miary F_1 podawana jest wartość odrzuceń.



Rysunek 6.10. Najlepsze wyniki klasyfikatorów w przeprowadzonych testach z włączonym progiem niepewności (numeracja: 1 – gry.pl, 2 – gsmonline.pl, 3 – kucharskie)

Wraz ze wzrostem liczby wstępnych komentarzy uczących klasyfikator DCM+ zwiększał obszar niepewności. Można powiedzieć, że im większą posiadał wiedzę, tym trudniej było mu zdecydować, czy komentarz jest poprawny, czy nie. Mimo większe pracy moderatora dla tego klasyfikatora, wzrost wstępnych komentarzy przyczyniał się do zmniejszenia błędu. Odwrotnie zachowywał się algorytm ICNN. Wzrost komentarzy wstępnego uczenia wzmagał pewność klasyfikatora co do wyboru kategorii, ale niestety nie przekładał się na poprawność odpowiedzi (wzrastał błąd). Pozostałe dwa klasyfikatory zachowywały się różnie w zależności od typu komentarzy.

6.2.3. Ocena wpływu uczenia inkrementacyjnego na ogólną jakość klasyfikacji

Kolejne wyniki dotyczą porównania sytuacji, w której moderator uczy klasyfikatory 200 komentarzami poprawnymi i 200 niepoprawnymi a następnie nie dokonuje modyfikacji bazy wiedzy w trakcie pracy klasyfikacyjnej. Wyniki porównane zostaną do sytuacji, w której wstępne uczenie stosuje po 20 komentarzy pozytywnych i negatywnych a algorytmy klasyfikacji uczą się inkrementacyjnie. Uzyskane wartości dotyczą średnich wyników z 5 testów dla wersji inkrementacyjnej (wersja tylko z uczeniem wstępnym zawsze daje te same wyniki, więc przeprowadzono tylko jeden test).

Tabela 6.2 przedstawia wyniki uzyskane po wyłączeniu odmawiania kategoryzacji. W tabeli litera W oznacza wynik po wyłączeniu uczenia inkrementacyjnego. Błąd został rozbity na błąd złego przydzielenia komentarza pozytywnego ($error_P$) i błąd złego przy-

dzielenia komentarza negatywnego ($error_N$). Zgodnie z wcześniejszymi wyjaśnieniami istotniejszy jest drugi z błędów.

Tabela 6.2. Wyniki dla włączonego (20 kom. wstępnych) i wyłączonego (200 kom. uczących) uczenia inkrementacyjnego przy wyłączonym progu niepewności dla komentarzy gsmonline.pl

Wynik/Algorytm	**Winnow**	**DCM+**	**k-NN**	**ICNN**
F_{1M}	0,645	0,644	0,647	0,640
F_{1MW}	0,231	0,688	0,616	0,628
$error_P$	0,174	0,184	0,182	0,188
$error_{PW}$	0,703	0,163	0,264	0,205
$error_N$	0,165	0,158	0,157	0,158
$error_{NW}$	0,005	0,108	0,097	0,125

Klasyfikator Winnow bardzo źle znosi brak uczenia inkrementacyjnego, gdyż zasada jego działania opiera się na uczeniu na podstawie błędów. Klasyfikator DCM+ znacznie lepiej radzi sobie bez uczenia inkrementacyjnego, o ile uzyska dużo komentarzy uczących na samym początku. Klasyfikatory k-NN i ICNN zachowują się podobnie. Co ciekawe, uzyskują w wersji bez uczenia inkrementacyjnego znacznie lepsze wyniki dla komentarzy błędnych niż dla poprawnych. Całościowo działają jednak podobnie do swych wersji inkrementacyjnych.

Tabela 6.3 przedstawia wyniki uzyskane w tych samych warunkach, ale po włączeniu progu niepewności.

Klasyfikator Winnow ponownie źle znosi brak uczenia inkrementacyjnego, ale co ciekawe w zasadzie nie myli się w przypadku komentarzy błędnych. Z drugiej strony często popełnia błąd przy komentarzach pozytywnych. Oznaczać to może chęć ciągłego odrzucania komentarzy poprawnych. Klasyfikator DCM+ osiągnął bardzo niskie wartości błędu, ale zostało to okupione ogromną liczbą odrzuceń klasyfikacji. Sytuację mogłaby poprawić zmiana domyślnego progu niepewności. Klasyfikatory k-NN i ICNN zachowują się podobnie. Także tu w wersji bez uczenia inkrementacyjnego uzyskują znacznie lepsze wyniki dla komentarzy błędnych niż dla poprawnych. Całościowo działają jednak podobnie do swych wersji inkrementacyjnych.

Tabela 6.3. Wyniki dla włączonego (20 kom. wstępnych) i wyłączonego (200 kom. uczących) uczenia inkrementacyjnego przy włączonym progu niepewności dla komentarzy gsmonline.pl

Wynik/Algorytm	**Winnow**	**DCM+**	**k-NN**	**ICNN**
unclass	0,268	0,300	0,204	0,220
unclass_{W}	0,127	0,840	0,186	0,106
error_{P}	0,082	0,077	0,110	0,106
error_{PW}	0,599	0,007	0,198	0,167
error_{N}	0,108	0,098	0,114	0,118
error_{NW}	0,000	0,012	0,066	0,111

6.2.4. Ocena wpływu uwzględniania nieznanych wyrazów na jakość klasyfikacji

Podstawowa wersja algorytmu przygotowania wektora słów na podstawie treści komentarza odrzuca nieznane wyrazy, których nie udało się poprawić korektą ortograficzną. Dwie kolejne tabele mają za zadanie odpowiedzieć na pytanie, czy uwzględnianie nieznanych wyrazów zmieni wyniki klasyfikatorów. Wszystkie przedstawiane wartości dotyczą średnich wyników z 5 przeprowadzonych testów.

Tabela 6.4 przedstawia wyniki uzyskane po wyłączeniu progu niepewności. W tabeli litera U oznacza wynik po włączeniu uwzględniania nieznanych wyrazów (nie stanowiących poprawnego lub możliwego do poprawienia słowa znajdującego się w alternatywnym słowniku ortograficznym). Jak poprzednio błąd został rozbity na błąd złego przydzielenia komentarza pozytywnego i błąd złego przydzielenia komentarza negatywnego.

Tabela 6.4. Wyniki dla włączonego i wyłączonego uwzględniania nieznanych wyrazów przy wyłączonym progu niepewności dla komentarzy gsmonline.pl (20 komentarzy wstępnych)

Wynik/Algorytm	**Winnow**	**DCM+**	**k-NN**	**ICNN**
F_{1M}	0,645	0,644	0,647	0,640
F_{1MU}	0,645	0,642	0,639	0,642
error_{P}	0,174	0,184	0,182	0,188
error_{PU}	0,174	0,183	0,184	0,184
error_{N}	0,165	0,158	0,157	0,158
error_{NU}	0,165	0,160	0,159	0,159

Tabela 6.5 przedstawia wyniki uzyskane po włączeniu progu niepewności.

Tabela 6.5. Wyniki dla włączonego i wyłączonego uwzględniania nieznanych wyrazów przy włączonym progu niepewności dla komentarzy gsmonline.pl (20 komentarzy wstępnych)

Wynik/Algorytm	**Winnow**	**DCM+**	**k-NN**	**ICNN**
unclass	0,268	0,300	0,204	0,220
$unclass_W$	0,259	0,304	0,201	0,218
$error_P$	0,082	0,077	0,110	0,106
$error_{PU}$	0,082	0,078	0,105	0,105
$error_N$	0,108	0,098	0,114	0,118
$error_{NU}$	0,110	0,092	0,119	0,117

We wszystkich algorytmach różnica jest minimalna, więc można powiedzieć, że uwzględnianie wyrazów, dla których nie udało się dokonać korekty ortograficznej, nie wpływa na jakość klasyfikacji. Do testów specjalnie został wybrany zestaw komentarzy *gsmonline.pl*, gdyż to w nim występuje dużo nazw własnych (nazwy producentów telefonów komórkowych, nazwy operatorów polskich i zachodnich, skróty: GSM, UMTS, EDGR, GPRS itp.). Brak różnicy wynika przede wszystkim z automatycznej korekty ortograficznej, która wiele nieznanych (choć poprawnych) nazw własnych zamienia na słowa ze słownika.

6.2.5. Średnie wartości jakości, błędu i odrzuceń dla poszczególnych typów komentarzy

Niniejszy podrozdział zawiera ostatnie zestawienie wyników klasyfikacji. Przedstawia średnie wartości (z 5 przeprowadzonych testów) uzyskane dla różnych typów komentarzy przez poszczególne klasyfikatory. Wszystkie dane zostały zebrane dla uczenia wstępnego korzystającego z 20 komentarzy pozytywnych i 20 negatywnych.

Tabela 6.6 przedstawia wyniki z wyłączonym progiem niepewności.

Tabela 6.6. Średnie wyniki klasyfikatorów z wyłączonym progiem niepewności (numeracja: 1 – gry.pl, 2 – gsmonline.pl, 3 – kucharskie). W każdym wierszu pogrubiona została wartość najlepsza

Wynik/Algorytm	**Winnow**	**DCM+**	**k-NN**	**ICNN**
F_{1M}-1	**0,730**	0,720	0,706	0,660
$error_P$-1	**0,160**	0,179	0,203	0,231
$error_N$-1	0,106	0,098	**0,090**	0,108
F_{1M}-2	0,645	0,644	**0,647**	0,640
$error_P$-2	**0,174**	0,184	0,182	0,188
$error_N$-2	0,165	0,158	**0,157**	0,158
F_{1M}-3	0,976	**0,981**	0,980	0,974
$error_P$-3	0,007	0,003	**0,000**	**0,000**
$error_N$-3	**0,016**	**0,016**	0,019	0,025

Różnice w wynikach poszczególnych algorytmów nie są duże. Pierwsze miejsce zajęły ex quo klasyfikatory Winnow i k-NN, które wygrały najwięcej razy, drugie miejsce zajął DCM+ a ostatnie ICNN. Jedyna znacząca różnica w wynikach pojawia się dla komentarzy *gry.pl* i algorytmu ICNN – okazał się on gorszy od konkurentów. Nie wiadomo, co było powodem takiego stanu rzeczy, bo w pozostałych dwóch uzyskiwał wyniki podobne do innych klasyfikatorów. Wartości przedstawione w tabeli są mało realne w rzeczywistych warunkach, bo w celu zmniejszenia błędu powinien być stosowany próg niepewności.

Tabela 6.7 przedstawia wyniki z włączonym odmawianiem komentarzy, czyli rezultaty zbliżone do tych, jakich można po algorytmach oczekiwać w momencie ich użycia w weryfikacji komentarzy związanych z w miarę konkretną dziedziną. Tabela przedstawia trzy wartości, których znaczenie zostało wyjaśnione na początku rozdziału. Najistotniejszy jest błąd przepuszczenia komentarzy niepoprawnych ($error_N$), następnie poziom odrzuceń (unclass) a na końcu błąd odrzucenia poprawnego komentarza ($error_P$).

Tabela 6.7. Średnie wyniki klasyfikatorów z włączonym progiem niepewności (numeracja: 1 – gry.pl, 2 – gsmonline.pl, 3 – kucharskie). W każdym wierszu pogrubiona została wartość najlepsza

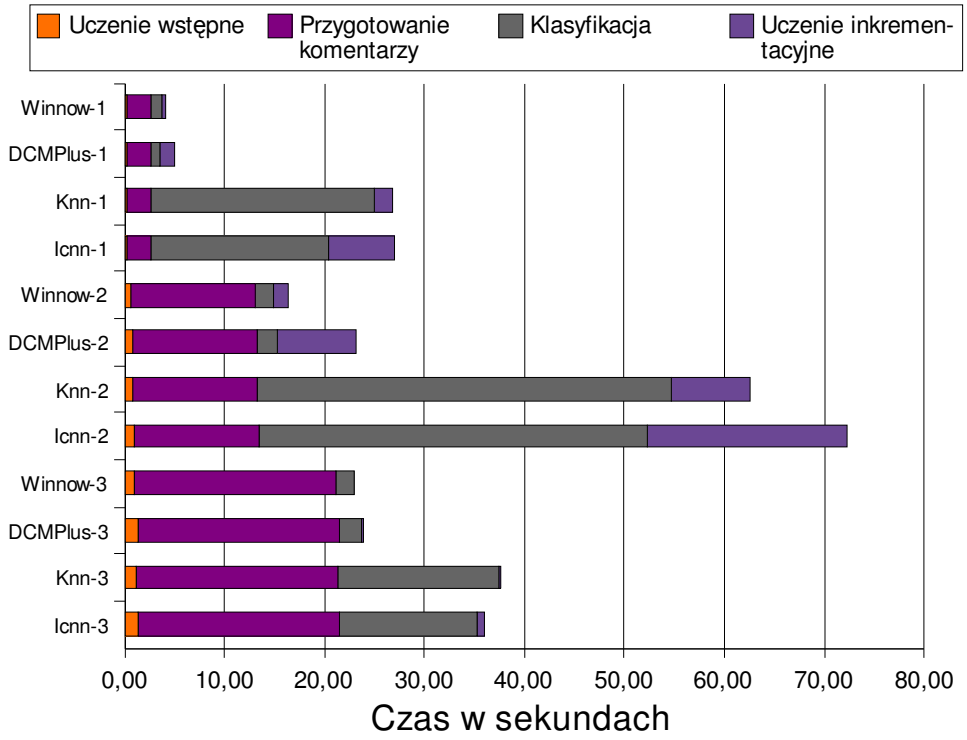
Wynik/Algorytm	**Winnow**	**DCM+**	**k-NN**	**ICNN**
unclass-1	0,265	0,396	0,259	**0,170**
$error_P$-1	0,084	**0,074**	0,125	0,178
$error_N$-1	0,049	**0,030**	0,059	0,082
unclass-2	0,268	0,300	**0,204**	0,220
$error_P$-2	0,082	**0,077**	0,110	0,106
$error_N$-2	0,108	**0,098**	0,114	0,118
unclass-3	0,072	0,101	0,043	**0,034**
$error_P$-3	0,002	**0,000**	**0,000**	**0,000**
$error_N$-3	**0,000**	**0,000**	0,002	0,006

Tym razem najlepszy w liczbie zwycięstw w poszczególnych wierszach okazał się algorytm DCM+, drugie miejsce zajął ICNN, trzecie k-NN a ostatnie Winnow. Klasyfikator DCM+ uzyskiwał najniższe wartości błędu, ale było to okupione największymi odmówieniami klasyfikacji w porównaniu z pozostałymi klasyfikatorami. Na drugim miejscu w kwestii liczby odrzuceń znalazł się klasyfikator Winnow. W miarę podobne i najniższe wartości odrzuceń uzyskały k-NN i ICNN. Zostało to jednak okupione największymi błędami klasyfikacji.

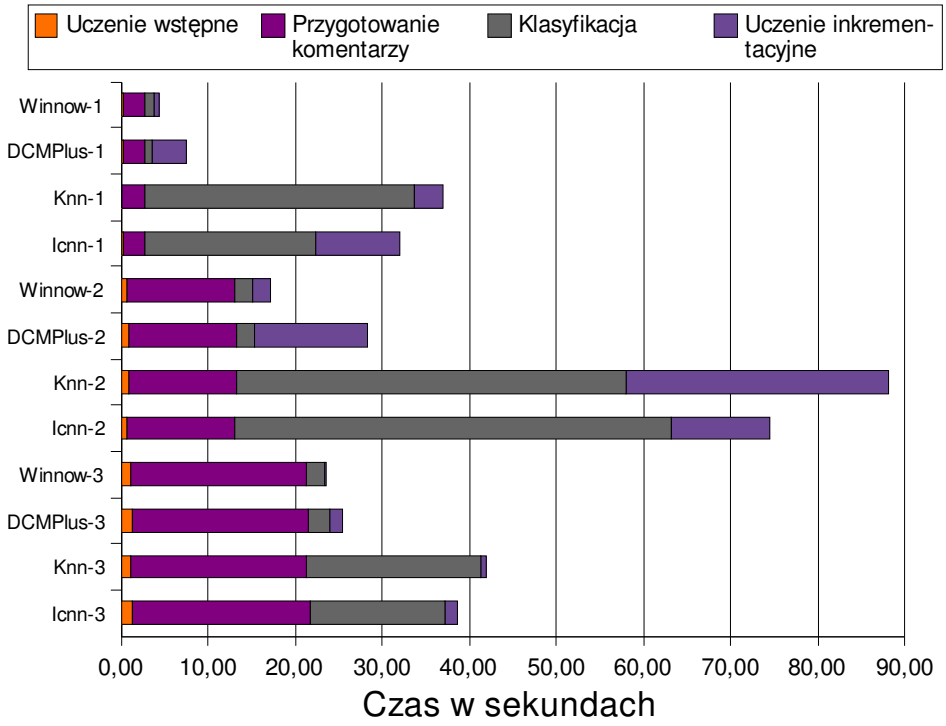
6.2.6. Całościowe wyniki czasowe weryfikacji komentarzy

Ostatnie zestawy wyników dotyczą sprawy bardzo istotnej z punktu widzenia administratorów i twórców witryn — czasu działania klasyfikacji oraz całej weryfikacji komentarzy. Przedstawione wyniki, jak wiele wcześniejszych, dotyczą uczenia wstępnego z użyciem 20 komentarzy pozytywnych i 20 negatywnych. Wyniki są średnią z 5 testów. Stanową sumę wykonań poszczególnych etapów klasyfikacji wszystkich dokumentów biorących udział w badaniach dla danego typu komentarzy.

Tradycyjnie już pierwszy wykres (rysunek 6.11) przedstawia wyniki dla wyłączonego progu niepewności natomiast drugi wykres (rysunek 6.12) dla włączonego progu.



Rysunek 6.11. Zsumowane średnie czasy wykonania poszczególnych etapów klasyfikacji przy wyłączonym progu niepewności (numeracja: 1 – gry.pl, 2 – gsmonline.pl, 3 – kucharskie)



Rysunek 6.12. Zsumowane średnie czasy wykonania poszczególnych etapów klasyfikacji przy włączonym progu niepewności (numeracja: 1 – gry.pl, 2 – gsmonline.pl, 3 – kucharskie)

Najmniejszy czas klasyfikacji posiadają w zasadzie ex quo klasyfikatory Winnow i DCM+. W rankingu czasu uczenia inkrementacyjnego (aktualizacji bazy wiedzy) wygrywa Winnow. Niewiele gorszy okazuje się DCM+. Klasyfikatory k-NN i ICNN mają bardzo duże czasy klasyfikacji i niejednokrotnie duże czasy aktualizacji. ICNN okazywał się w większości przypadków szybszy od k-NN na etapie klasyfikacji, ale wolniejszy w uczeniu inkrementacyjnym.

Włączenie progu niepewności powoduje zmniejszenie szybkości działania klasyfikatorów k-NN i ICNN zarówno przy klasyfikacji, jak i aktualizacji danych. Pozostałe klasyfikatory próg spowalniał tylko z racji częstszych aktualizacji. Jedynym wyjątkiem jest klasyfikator ICNN dla drugiego zestawu komentarzy (*gsmonline.pl*), w którym to próg niepewności zmniejszył liczbę i czas trwania aktualizacji.

6.3. WNIOSKI Z WYNIKÓW

Na podstawie przedstawionych tabel i wykresów wysnuć można wiele interesujących wniosków. Zostaną one przedstawione w kolejnych podrozdziałach w rozbiciu na poszczególne algorytmy klasyfikacji. Przedostatni podrozdział przedstawia i motywuje wybór klasyfikatora (z czterech testowanych) zalecanego do przeprowadzania weryfikacji komentarzy pojawiających się na rzeczywistych witrynach. Ostatni podrozdział opisze wyniki z perspektywy rodzajów komentarzy.

6.3.1. Balanced Winnow

Klasyfikator ten okazał się bardzo szybki w trakcie klasyfikacji oraz uczenia inkrementacyjnego. W zasadzie po połączeniu obu tych wartości okazywał się najszybszy. Jakość klasyfikacji umiejscawia go w środku stawki przy stosowaniu odmawiania kategoryzacji niektórych komentarzy i pozwala czasem wygrywać, gdy odmawianie jest wyłączone. Jest jedynym algorytmem, który notował stabilne zmniejszanie błędu i odmówień wraz z nadchodzeniem nowych danych dzięki uczeniu inkrementacyjnemu. Bardzo słabo spisywał się, gdyby uczenie inkrementacyjne nie miałoby być przeprowadzane lub byłoby wykonywane rzadko.

Balanced Winnow nadaje się do dynamicznej klasyfikacji tekstów na stronach WWW mimo, że jego wyniki nie pozwolą czuć się moderatorowi całkowicie pewnie.

6.3.2. DCM+

To drugi pod względem szybkości klasyfikator. Tylko w niewielkim stopniu ustępuje najlepszemu, a od następnych dzieli go ogromna różnica wydajności. Stosunkowo dobrze reaguje na zwiększenie liczby komentarzy uczenia wstępnego, choć zmniejszeniu błędu towarzyszy większa niepewność (zwiększa się liczba odmówień kategoryzacji). Daje dobre wyniki przy wyłączonym uczeniu inkrementacyjnemu w przypadku wyłączenia odmówień. W zasadzie są one lepsze od tych z wersji inkrementacyjnej. Wyłączenie uczenia inkrementacyjnego i włączenie odmawiania w ogromny sposób zwiększa niepewność klasyfikatora, ale skutkuje minimalnym błędem. Jest to raczej dobra cecha niż wada, gdyż klasyfikator nie dopuści do przedostania się niepoprawnych tekstów, jeśli moderator rzadko uaktualnia bazę wiedzy. Zachowa komentarze do sprawdzenia. W wersji z uczeniem inkrementacyjnym daje najmniejsze błędy choć okupione jest to dużą niepewnością.

Z przedstawionego opisu wynika, że algorytm ten w niemal naturalny sposób nadaje się do dynamicznej klasyfikacji w internecie, bo zapewnia najmniejsze błędy, ale niestety jest to związane z dosyć dużą liczbą odmówień klasyfikacji. Algorytm idealnie nadaje się dla moderatorów chcących uzyskać wysoki standard przy nie tak imponującym jak przy innych algorytmach klasyfikacji zmniejszeniu nakładu pracy własnej.

6.3.3. k-NN

W porównaniu z dwoma wcześniej opisanymi algorytmami trudniej dostrzec w tym klasyfikatorze poprawę wyników w kolejnych etapach inkrementacyjnego uczenia. Niemniej one występują. Klasyfikator k-NN gorzej radzi sobie z brakiem aktualizacji bazy wiedzy, choć w takiej sytuacji znacząco zmniejsza się liczba błędów z zatwierdzeniem niepoprawnych komentarzy, co jest istotne. Z drugiej strony zwiększa się liczba błędów dla komentarzy pozytywnych. Są one jednak mniej ważne. Klasyfikator okazuje się dobry w klasyfikacji bez odmawiania, ale znacznie gorzej radzi sobie, gdy musi stosować próg niepewności. Czas jego działania okazał się najwyższy wśród wszystkich testowanych algorytmów, co nie powinno dziwić z racji ogromu komentarzy testowanych na etapie klasyfikacji.

Algorytmu tego nie warto stosować w dynamicznej klasyfikacji, gdyż nie zapewnia od dobrych wyników dla wersji z odmawianiem klasyfikacji a czas jego działania jest ogromny w porównaniu z niektórymi konkurencyjnymi rozwiązaniami.

6.3.4. ICNN

Algorytm ten niestety nie spełnił pokładanych w nim nadziei w związku z nie w pełni liniową separacją komentarzy pozytywnych i negatywnych. Klasyfikator o liniowej separacji poradził sobie lepiej. Czasy działania algorytmu są niższe od k-NN z racji łączenia dokumentów w grupy. Niestety jednocześnie są znacząco wyższe od pozostałych dwóch klasyfikatorów. W oczy rzuca się brak na wykresach jakichkolwiek oznak zmniejszania błędu i niepewności wraz z pojawianiem się inkrementacyjnie nowych danych. Algorytm jest więc w tym względzie raczej mało stabilny. Wzrost liczby komentarzy uczenia wstępnego zamiast zwiększać jakość klasyfikacji, zmniejsza ją. Klasyfikator poprawia swój błąd dotyczący niepoprawnych komentarzy, ale zwiększa przy tym błąd dla komentarzy poprawnych. Do zalet zaliczyć można minimalny obszar niepewności (w dwóch rodzajach komentarzy okazał się najmniejszy), choć było to okupione znacznym błędem klasyfikacji.

Podobnie jak k-NN, również tego algorytmu nie warto stosować do klasyfikacji online, przede wszystkim z racji dużego czasu działania i średnich osiągów.

6.3.5. Najlepszy z czterech testowanych klasyfikatorów

Po przeprowadzeniu szeregu testów dla różnych możliwe jest wskazanie zwycięzców nadających się do klasyfikacji dynamicznej na stronach WWW.

Klasyfikator DCM+ doskonale nadaje się do weryfikacji komentarzy z zastosowaniem możliwości odmówienia kategoryzacji. Uzyskuje wtedy minimalny błąd. Jest szybki i stosunkowo prosty w implementacji. Poza ustawieniem progu niepewności nie wymaga od użytkownika strojenia żadnych dodatkowych parametrów (czego wymagają pozostałe trzy algorytmy).

Gdyby z jakichś powodu potrzebny był algorytm weryfikacji z uczeniem inkrementacyjnym, ale bez możliwości odmawiania, warto zastanowić się nad algorytmem Balanced Winnow, który w tej sytuacji daje wyniki lepsze niż DCM+.

6.3.6. Wyniki a rodzaj komentarzy

Z przedstawionych wcześniej informacji jednoznacznie wynika, iż stworzony sztucznie zestaw komentarzy przepisy kucharskie kontra komentarze poprawne z *gsmonline.pl* nie sprawiał wszystkim klasyfikatorom najmniejszych problemów. Separacja obu dziedzin okazywała się tak duża, że pomyłki zdarzały się rzadko. Dziwi jedynie fakt, że w tej dużej separacji lepiej radziły sobie algorytmy, które teoretycznie powinny dobrze radzić sobie w

nieliniowej separacji. Algorytmy klasyfikacji dla liniowej separacji okazywały się gorsze. Przypuszcza się, iż wynika to z większej czułości algorytmów nieliniowej separacji na szum, który w tym sztucznym zestawie był niewielki, co pozwalało osiągnąć lepsze rezultaty.

Komentarze z witryny *gry.pl* są stosunkowo krótkie, ale mimo to algorytmy radziły sobie z nimi lepiej niż w przypadku długich komentarzy *gsmonline.pl*. Sadzi się, że wynika to z mniejszych niejednoznaczności — mniej wyrazów, mniej możliwości powiedzenia tego samego na różne sposoby. Ogólnie jednak uzyskiwana jakość klasyfikacji rozczarowuje w większości algorytmów. Jedynie algorytm DCM+ w niektórych testach potrafił zejść z błędem ogólnym poniżej 10%, ale było to okupione 30% komentarzy z odmówioną klasyfikacją. Gdyby porównać wyniki z wyłączonym odmawianiem obraz jest jeszcze gorszy. Najlepszy w tym przypadku algorytm Winnow uzyskiwał błąd ogólny na poziomie 26%.

Z poprzedniego akapitu wynika, że stosowanie progu niepewności jest w weryfikacji komentarzy nie przydatną opcją, ale koniecznością. Na duży błąd uzyskiwany w realnym zbiorze komentarzy wpływ ma również automatyczne przygotowanie wektorów wag a w szczególności korekta ortograficzna mogąca wprowadzać znaczny błąd. Niestety jest to niezbędny element, który na ogół w innych rodzajach klasyfikacji tekstów w ogóle nie występuje, bo ma się pewność co do poprawności analizowanych teksów. Komentarze od dowolnych użytkowników witryny w żaden sposób tego nie gwarantują.

Zbiór komentarzy *gsmonline.pl* uzyskał najgorsze wyniki – odpowiednio w najlepszym algorytmie błąd wyniósł 33% (przy wyłączonym progu) i 17% (przy włączonym progu). Wynika to z faktu, iż poza problemami opisanymi w poprzednim akapicie dochodziło jeszcze do wielu niuansów powodujących odrzucanie części komentarzy. Niektóre komentarze były w znaczniej mierze poprawne i tylko niewielki fragment powodował ich odrzucenie przez moderatora. Algorytmy klasyfikacji mają w takiej sytuacji naprawdę ciężki orzech do zgryzienia, gdyż korzystają ze statystycznych rozkładów wyrazów. Wiedza o sposobie działania klasyfikatora umożliwia niejednokrotnie jego oszukanie.

Za niezbyt dobre ogólne wyniki klasyfikatorów wpływ mógł mieć również podział komentarzy na poprawne i niepoprawne wykonywany przez autora pracy na podstawie komentarzy istniejących na witrynach. Najbardziej rażące komentarze w ogóle nie pojawiły się w testach z powodu ich wcześniejszego usunięcia przez administratorów witryn. Wnioskować można, że w rzeczywistości po zastosowaniu wszystkich opisanych mechani-

zmów wykrywania wulgaryzmów, adresów WWW itp. rzeczywista jakość klasyfikacji przedstawianych algorytmów będzie większa.

7. PODSUMOWANIE

Niniejsze rozdział podsumowuje prace implementacyjne i badawcze wykonane na potrzeby pracy dyplomowej magisterskiej. Dodatkowo formuje ogólne wnioski płynące z prób zastosowania automatycznej weryfikacji komentarzy (klasyfikacji tekstów) na stronach WWW dla języka polskiego przy zastosowaniu języka skryptowego PHP i bazy danych MySQL. Są one o tyle istotne, że zgodnie z wiedzą autora stanowią jedną z pierwszych w Polsce prób prowadzenia tego rodzaju automatycznej weryfikacji. Ostatni podrozdział przedstawia tematy, którymi w pracy się nie zajęto a mogą wpłynąć na poprawę lub przyspieszenie klasyfikacji tekstów.

7.1. ZREALIZOWANE ZADANIA I TESTY

7.1.1. Przygotowanie komentarzy

Ponieważ nie istnieje ogólnie dostępna baza komentarzy w języku polskim, trzeba było ją stworzyć. Na podstawie danych zebranych z trzech różnych witryn powstały trzy zbiory komentarzy. Autor pracy dokonał ich ręcznej klasyfikacji, co pozwoliło uzyskać ponad 2400 sklasyfikowanych komentarzy do prowadzenia testów. Podejście to miało jednak pewną wadę polegającą na braku bardzo mocno negatywnych komentarzy, gdyż zostały one już wcześniej usunięte przez moderatorów witryn. Warto podkreślić zachowanie oryginalnej postaci komentarzy — zawierają one błędy ortograficzne, często nie mają znaków diakrytycznych, bogata jest ich interpunkcja itp.

Po tym etapie zajęto się próbą możliwie dobrego przekształcenia komentarzy na wektory cech (słów) wykorzystywane przez algorytmy klasyfikacji tekstów. Wykonano specjalne wyrażenia regularne wychwytujące nie tylko poprawne formalnie, ale również niektóre specjalnie źle napisane adresy e-mail i URL. Na potrzeby pracy powstał specjalny, autorski algorytm tokenizacji wykrywający między innymi powtórzenia znaków i próby ukrycia wyrazów wulgarnych. Dodatkowo algorytm zamienia liczby na kilka wartości przedziałowych.

Na podstawie alternatywnego słownika ortograficznego, programu budującego automaty FSA i kilku własnych skryptów przekształcających wykonano automaty FSA dla korekty ortograficznej, diakrytycznej oraz lematyzacji. Dodatkowo przygotowano listę wyrazów wulgarnych i tzw. stoplistę, czyli listę wyrazów usuwanych z dalszych etapów klasyfikacji.

Wzorując się na kodzie w języku C autorstwa Jana Daciuka, od podstaw zaimplementowano w PHP algorytm korekty ortograficznej i diakrytycznej z wykorzystaniem wykonanych automatów. W celu przeprowadzenia lematyzacji (sprowadzania wyrazu do wersji słownikowej) wykorzystano algorytm autorstwa Wojciecha Rutkowskiego (oparty na pracach Jana Daciuka), dostosowując go do języka PHP 5 i innych wartości adresowych w formacie zapisu automatu skończonego.

7.1.2. Algorytmy klasyfikacji tekstów

Przeanalizowano wiele publikacji naukowych dotyczących klasyfikacji tekstów, by poznać stosowane dawniej i obecnie metody klasyfikacji. W pracy pokrótce przedstawiono charakterystyki kilkunastu z nich. Zestawiono wyniki przedstawiane w publikacjach, by określić względna jakość klasyfikatorów. Na podstawie zebranych informacji wybrano do implementacji cztery algorytmy, które charakteryzowały się dobrą jakością i umożliwiały (lub po przeróbkach mogłyby zapewnić) uczenie inkrementacyjne.

Wszystkie cztery algorytmy zostały w całości zaimplementowane w PHP na podstawie wzorów i opisów znajdujących się w publikacjach naukowych. Klasyfikatory Balanced Winnow i DCM+ zostały zaimplementowane bez żadnych znaczących modyfikacji. Dokonano jedynie ich optymalizacji, wykorzystując fakt, iż weryfikacja komentarzy jest klasyfikacją binarną.

Klasyfikator k-NN, stosowany powszechnie w wielu testach, został wykonany z wykorzystaniem kilku modyfikacji (inne modyfikatory wag słów, inna miara podobieństwa, wprowadzenie usuwania starych i niebiorących udziału w klasyfikacji dokumentów na zasadzie LRU). Czwarty z implementowanych algorytmów (ICNN) był nową propozycją autora pracy rozbudowującą algorytm kNN (brak łącznika ma znaczenie) o uczenie inkrementacyjne a także o rozwiązania użyte w klasyfikatorze k-NN (modyfikatory wag, miara podobieństwa itp.).

Do wszystkich algorytmów wprowadzono stosunkowo rzadko stosowany w innych pracach próg niepewności, czyli odmawianie kategoryzacji dokumentu, jeśli różnica ocen między kategorią pozytywną i negatywną jest mniejsza od zadanej wartości. Zastosowanie progu zmniejsza błąd klasyfikacji, ale wymaga większego nakładu pracy moderatora.

7.1.3. Testy

Na wykonanych algorytmach przygotowywania i klasyfikacji komentarzy przeprowadzono szereg testów. W przygotowaniu wstępnym skupiono się przede wszystkim na szybkości działania zaimplementowanych rozwiązań z ewentualnymi, konkurencyjnymi algorytmami.

Przed przystąpieniem do głównych testów klasyfikatorów wykonano szereg krótszych testów mających ustalić najbardziej optymalne parametry klasyfikatorów dla poszczególnych rodzajów komentarzy.

Klasyfikatory testowano zarówno pod kątem szybkości, jak i jakości uzyskiwanych wyników. Niemalże wszystkie testy wykonano dwukrotnie — z włączonym i wyłączonym progiem niepewności. Sprawdzono wpływ uczenia inkrementacyjnego i wstępnego na poszczególne algorytmy oraz uwzględnianie przy klasyfikacji nierozpoznanych wyrazów. Przedstawiono najlepsze i średnie wyniki uzyskane w testach przez klasyfikatory. Wykonano też pomiary szybkości działania poszczególnych faz weryfikacji.

7.2. OGÓLNE WNIOSKI Z UZYSKANYCH WYNIKÓW

Po przedstawieniu w poprzednim rozdziale wyników badań dla czterech algorytmów można spróbować wyciągnąć wnioski co do ogólnej możliwości stosowania automatycznej weryfikacji komentarzy na stronach WWW.

Dwa z przedstawionych algorytmów klasyfikacji (Balanced Winnow i DCM+) okazały się na tyle szybkie, że z powodzeniem uda się je zastosować na stosunkowo mocno obciążonych serwisach internetowych. Czas klasyfikacji i uczenia inkrementacyjnego dla 800 komentarzy nie przekraczał najczęściej kilku sekund. Można założyć, że na szybkim procesorze jednordzeniowym wykorzystującym język PHP 5.1 i bazę danych MySQL algorytmy te mogą przetwarzać do 200 wektorów słów na sekundę. Pozostałe dwa algorytmy (k-NN i ICNN) choć niejednokrotnie osiągały wyniki jakościowe podobne do dwóch wcześniej wymienionych, działały znacznie dłużej i wymagały więcej miejsca w bazie danych.

Niejednokrotnie czas właściwej klasyfikacji okazywał się krótszy od czasu przygotowania do niej komentarza. Największy udział w czasie przygotowywania miała korekta ortograficzna i sprowadzanie słów z formy odmienionej do podstawowej (lematyzacja) i to pomimo zastosowania szybkich algorytmów. Z elementów tych nie należy rezygnować, gdyż zapewniają polepszenie jakości klasyfikacji przez znoszenie fleksyjności języka pol-

skiego i próbę poprawiania błędów ortograficznych i uzupełniania diakrytyki, która niezwykle często zdarza się w komentarzach nadsyłanych przez internautów.

Warto zadawać sobie sprawę, że algorytmy klasyfikacji testowane były odmiennie niż w wielu innych publikacjach. Uczenie wstępne zawierało do kilkudziesięciu komentarzy (bo mniej więcej taką liczbą dysponuje początkowo moderator nowej witryny). Algorytmy zaraz po uczeniu wstępnym przystępowały do testów, w których jednocześnie uczyły się na własnych błędach. Wykresy z poprzedniego rozdziału pozwalają przypuszczać, iż wyniki po dłuższym czasie działania (mierzone na przykład w zakresie od tysiąca do dwóch tysięcy komentarzy bez uwzględniania pierwszego tysiąca) mogą być znacznie lepsze od zaprezentowanych w niniejszej pracy.

Uzyskane wyniki jakościowe (nawet najlepszego algorytmu) nie zachwyciły, ale i nie przekreśliły możliwości stosowania automatycznej weryfikacji komentarzy. Najlepszy w testach z odmawianiem kategoryzacji niepewnych komentarzy algorytm (DCM+) dopuszczał wyświetlenie niepoprawnego komentarza średnio (dla wszystkich zbiorów testowych) w 7% a moderator musiał dodatkowo ocenić poprawność około 30% nadchodzących tekstów. Gdy wyłączono dopuszczanie odmawiania, błąd wyświetlenia niepoprawnego komentarza rósł dwu- lub trzykrotnie. Co ważne, testy wykazały, że jeśli separacja dziedzinowa komentarzy poprawnych i błędnych będzie bardzo dobra (komentarze poprawne będą dotyczyły wyjątkowo ścisłej tematyki), możliwe jest uzyskanie zerowego błędu wyświetlania niepoprawnego komentarza przy kilku procentach odmówionych kategoryzacji.

Automatyczna weryfikacja komentarzy przedstawiona i zaimplementowana w niniejszej nadaje się do natychmiastowego wdrożenia na wielu witrynach korzystających z języka skryptowego PHP 5.1 lub nowszego. Pozwala przynajmniej o 60% zredukować nakład pracy moderatorów poświęcany na weryfikację komentarzy. Szczególnie warto polecić ją witrynom, których moderatorzy stosunkowo rzadko mają możliwość zatwierdzania poszczególnych komentarzy lub usuwania tylko niepoprawnych. Dzięki niej najbardziej niepoprawne komentarze zostaną odrzucone automatycznie bez dodatkowej ingerencji.

7.3. ZADANIA NA PRZYSZŁOŚĆ

Pomimo poruszenia w niniejszej pracy wielu zadań i dużej staranności o zachowanie możliwie wysokiej jakości weryfikacji komentarzy, z powodu ograniczeń czasowych zre-

zygnowano z rozwinięcia kilku tematów mogących poprawić szybkość lub też jakość klasyfikacji. Poleca się je uwadze osób chcących rozwinąć prezentowane tematy.

W testach język PHP okazał się około 100-krotnie wolniejszy od języka C. Warto byłoby się zastanowić na wykonaniem modułów PHP wykorzystujących w sposób natywny bibliotekę FSA do korekty ortograficznej, diakrytycznej oraz lematyzacji. Pozwoliłoby to znacząco przyspieszyć fazę przygotowania komentarzy. Można także zastanowić się nad algorytmem rozwiązywania niejednoznaczności w korekcie ortograficznej, na przykład proponując moderatorom określenie do kilkuset wyrazów, które są często błędnie wpisywane na danym forum.

Warto byłoby również przeprowadzić testy z użyciem innych miar podobieństwa wektorów wag słów oraz modyfikatorów wag, gdyż w innych publikacjach wykazano znaczny wpływ tych rozwiązań na końcowe wyniki. Poprawienie czasu właściwej klasyfikacji w porównaniu z prezentowanymi w niniejszej pracy wynikami nie będzie łatwe (bo niewiele istniejących algorytmów jest równie prostych do szybkiej implementacji z wykorzystaniem bazy danych jak Balanced Winnow). Warto skupić się na poprawie jakość, gdyż to ona jest główną bolączką przedstawionych rozwiązań.

LITERATURA

[1] Aas K., Eikvil L.: *Text categorisation: A survey*, Raport techniczny, Norwegian Computing Center, 1999

[2] Achour M. i in.: Podręcznik PHP, *http://www.php.net/manual/pl/*

[3] Alternatywny słownik ortograficzny dla języka polskiego, *http://www.kurnik.pl/slownik/ort/*

[4] Bergo A.: *Text Categorization and Prototypes*, 2001, *http://www.illc.uva.nl/Publications/ResearchReports/MoL-2001-08.text.pdf*

[5] Bloehdorn S., Hotho A.: *Text classification by boosting weak learners based on terms and concepts*, In Proceedings of the 4th IEEE International Conference on Data Mining (ICDM), 2004, s. 331–334

[6] Borycki Ł., Sołdacki P.: *Automatyczna klasyfikacja tekstów*, III Krajowa Konferencja MiSSI 2002, Kliczków, 2002, *http://www.zsi.pwr.wroc.pl/zsi/missi2002/pdf/s504.pdf*

[7] Ciura M., Deorowicz S.: *How to squeeze a lexicon*, Software — Practice & Experience 2001, 31 (11), s. 1077–1090

[8] Ciura M., Deorowicz S.: *Correcting spelling errors by modelling their causes*, International Journal of Applied Mathematics and Computer Science, 2005, 15(2), s. 275–285

[9] Daciuk J.: *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing*, rozprawa doktorska, Politechnika Gdańska, 1998, *http://www.pg.gda.pl/~jandac/thesis.ps.gz*

[10] Daciuk. J.: *FSA — automaty proste*, programy w języku C, *http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/fsa_polski.html*

[11] Dagan I. i in.: *Mistake-Driven Learning in Text Categorization*, Proceedings of EMNLP-97, 2nd Conference on Empirical Methods in Natural Language Processing, 1997

[12] Friedl J.: *Wyrażenia regularne*, Wydawnictwo Helion, Gliwice, 2001

[13] Fung, G.P.C. i in.: *Discriminative category matching: efficient text classification for huge document collections*, Data Mining 2002. ICDM 2002. Proceedings of 2002 IEEE International Conference, 2002, s. 187–194

[14] Guo G. i in.: *Using kNN Model-based Approach for Automatic Text Categorization*, Soft Computing Journal, 2003

[15] Han E. i Karypis G.: *Centroid-Based Document Classification: Analysis and Experimental Results*, Lecture Notes In Computer Science, wol. 1910, Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, 2000, s. 424–431

[16] Han E. i in.: *Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification*, Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2001, s. 53–65

[17] Joachims T.: *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*, Technical report (LS VIII-Report), Uniwersytet Dortmundzki, Niemcy, 1997

[18] Joachims T.: *Text categorization with support vector machines: Learning with many relevant features*, Proceedings of the Tenth European Conference on Machine Learning (ECML-98), 1998, s. 137–142

[20] Koster C.: *Chapter 2. Document Classification*, Lecture Notes, 2003, *http://www.cs.ru.nl/~kees/ir2/papers/h03.pdf*

[21] Lee K.: *Text Categorization with a Small Number of Labeled Training Examples*, rozprawa doktorska, Uniwersytet Sydney, Australia, 2003

[22] Michelakis E. i in.: *Filtron: A Learning-Based Anti-Spam Filter*, Uniwersytet Ateński, Grecja, 2004

[23] MySQL AB: *MySQL 5 Reference Manual*, 2006, *http://dev.mysql.com/doc/refman/5.0/en/index.html*

[24] Nagórko A.: *Zarys gramatyki polskiej*, PWN, Warszawa 2003

[25] Oflazer K.: *Error-tolerant finite state recognition with applications to morphological analysis and spelling correction*, Computational Linguistics 1996, 22(1), s. 73–89

[26] Poncon G.: *PHP. Hobby, które przynosi zyski*, PHP Solutions, 2006, nr 2 (13), s. 80

[27] Porter M.: *An algorithm for suffix stripping*, program, 1980, 14(3), s. 130–137

[28] PWN: *Lista słów (słownik frekwencyjny na podstawie wersji demonstracyjnej korpusu PWN)*, Witryna PWN, 2006, *http://korpus.pwn.pl/stslow.php*

[29] Rutkowski W.: *Lematyzator w języku PHP*, Projekt SENECA, 2005, *http://seneca.kie.ae.poznan.pl*

[30] Sebastiani F.: *A Tutorial on Automated Text Categorisation*, Proceedings of ASAI-99, 1st Argentinian Symposium on Artificial Intelligence, Buenos Aires, Argentyna, 1999, s. 7–35

[31] Sebastiani F.: *Machine learning in automated text categorization*, ACM Computing Surveys, 34(1), 2002, s. 1–47

[32] Siefkes C. i in.: *Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering*, Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004), s. 410–421

[33] Strychowski J.: *Zastosowanie metod uczenia maszynowego w analizie morfologicznej języka polskiego*, Zarządzanie i technologie informacyjne T2. Metody sztucznej inteligencji w zarządzaniu i sterowaniu, Wydawnictwo Uniwersytetu Śląskiego, 2004

[34] Suszczańska N., Lubińska M.: *POLMORPH, Polish Language Morphological Analysis Tool*, 19th IASTED International Conference Applied Informatics-AL'2001, Innsbruck 2001, s. 6

[35] Weiss D.: *A Survey of Freely Available Polish Stemmers and Evaluation of Their Applicability in Information Retrieval*, 2nd Language and Technology Conference, Poznań 2005, s. 216–221

[36] Weiss D.: *Lemetyzator w języku Java*, *http://www.cs.put.poznan.pl/dweiss/xml/projects/lametyzator/index.xml*

[37] Wilk S.: *Zastosowanie naiwnego klasyfikatora Bayes'a do klasyfikowania dokumentów tekstowych*, *http://www-idss.cs.put.poznan.pl/~krawiec/snumlab/lab5teoria.htm*

[38] Witryna GSMonline.pl: *http://www.gsmonline.pl*

[39] Witryna programu Aspell, *http://aspell.net*

ZAŁĄCZNIKI

Płyta CD-ROM zawierająca:

- treść niniejszej pracy dyplomowej magisterskiej w formacie PDF — folder *praca*),
- kod źródłowy wszystkich algorytmów wykonanych na potrzeby pracy (wraz z komentarzem) — folder *kod*,
- słowniki do lematyzacji i korekty ortograficznej w formacie FSA przygotowane na potrzeby pracy — folder *slowniki*,
- strukturę bazy danych wraz z treścią komentarzy używanych w trakcie testów — folder *baza*,
- surowe wyniki z przeprowadzonych testów — folder *testy*.