|  |  |
| --- | --- |
| Politechnika Warszawska  Wydział Elektroniki i Technik Informacyjnych  Instytut Informatyki | Rok akademicki 2012/2013 |



praca dyplomowa magisterska

Adrian Wiśniewski

Inteligentne wykrywanie reklam w sieci WWW na podstawie adresów URL

Opiekun pracy

prof. dr hab. inż. Mieczysław Muraszkiewicz

|  |  |
| --- | --- |
| Ocena |  |
| Podpis przewodniczącego  Komisji Egzaminu Dyplomowego |  |

|  |  |
| --- | --- |
|  | Kierunek: Informatyka  Specjalność: Inżynieria Systemów Informatycznych  Data urodzenia: 24 grudnia 1987 r.  Data rozpoczęcia studiów: październik 2006 r. |

Życiorys

Urodziłem się 24 grudnia 1987 r. w Nowym Dworze Mazowieckim. Mieszkam w Wołominie, gdzie ukończyłem Sportową Szkołę Podstawową nr 5 im. C. K. Norwida i Sportowe Gimnazjum nr 3 im. Polskich Olimpijczyków. Następnie uczęszczałem do klasy o profilu matematyczno-fizycznym w IV Liceum Ogólnokształcącym im. Adama Mickiewicza w Warszawie. W październiku 2006 r. rozpocząłem studia na Politechnice Warszawskiej na Wydziale Elektroniki i Technik Informacyjnych na kierunku Informatyka.

Rok później zacząłem pracować jako programista aplikacji internetowych oraz mobilnych. Przez kilka lat wykonywałem różne zlecenia dla klientów takich jak McDonald’s, Allegro, czy gazeta.pl. Mimo wielu sukcesów, postanowiłem – zgodnie ze swoimi marzeniami z dzieciństwa – tworzyć gry komputerowe i w sierpniu 2011 r. zatrudniłem się w firmie 11 bit studios S.A. Przez dwa lata – do chwili obecnej – pracowałem nad grą Anomaly 2, dodając do silnika gry możliwość rozgrywki wieloosobowej przez sieć. Po zakończeniu studiów zamierzam przeprowadzić się do Stanów Zjednoczonych Ameryki i podjąć pracę w firmie Google, gdzie będę mógł rozwijać się w dziedzinie sztucznej inteligencji.

Od 2006 roku należę do społeczności twórców gier komputerowych www.gamedev.pl, a w latach 2009 – 2010 pełniłem rolę administratora i członka zarządu Koła Naukowego Twórców Gier Komputerowych „Polygon”.

|  |  |
| --- | --- |
|  | Podpis studenta |

Egzamin dyplomowy

Złożył egzamin dyplomowy dn. 2013 r.

Z wynikiem

Ogólny wynik studiów

Dodatkowe wnioski i uwagi Komisji

Streszczenie

Rozwój sieci WWW pociągał za sobą rozwój branży reklamowej. Materiały promocyjne można zobaczyć na prawie każdej stronie internetowej, a ich natężenie bywa przytłaczające – szczególnie dla posiadaczy urządzeń mobilnych, którzy płacą za pobierane dane. Moja praca magisterska zawiera opis systemu wykrywającego i blokującego reklamy w sieci WWW na podstawie adresów URL bez konieczności ich pobierania. Proponowane rozwiązanie wykorzystuje metody sztucznej inteligencji i uczenia maszynowego, aby usprawnić i w pełni zautomatyzować proces tworzenia reguł rozróżniających reklamy od treści. Obecnie funkcjonujące programy używają specjalnych wyrażeń regularnych, których lista musi być stale i na bieżąco uaktualniania przez administratorów. Powoduje to powstanie dużych kosztów utrzymania systemu, które można wyeliminować poprzez zastosowanie metod inteligentnych.

*Słowa kluczowe:*

Wykrywanie reklam, Sztuczna inteligencja, Uczenie maszynowe, Eksploracja danych

Intelligent advertisement detection in WWW based on URL addresses.

Development of World Wide Web has caused growth of the advertising industry. Advertisements are present on almost every web page and often their intensity is overwhelming for the user, especially if he has a mobile device and pays for every single byte of downloaded data. My thesis contains description of a system, that detects and blocks ads in WWW based on their URL addresses without needing to download them. Proposed solution uses methods of artificial intelligence and machine learning to improve and fully automatize creation of rules, that discriminate between ads and legitimate content. Current products use special kind of regular expressions. Their list must be constantly updated on a regular basis. This results in high costs of system maintenance, which can be eliminated with use of intelligent methods.

*Keywords:*

Ad blocking, Ad detection, Artificial intelligence, Machine learning, Data mining

Spis treści

[Rozdział 1. Wstęp 6](#_Toc359171427)

[1.1. Cel i zakres pracy 6](#_Toc359171428)

[1.2. Powiązane prace naukowe i publikacje 7](#_Toc359171429)

[Rozdział 2. Filtrowanie reklam w sieci WWW 8](#_Toc359171430)

[2.1. Po czym rozpoznać reklamę? 9](#_Toc359171431)

[2.2. Obecne rozwiązania 11](#_Toc359171432)

[2.3. Metody inteligentne 12](#_Toc359171433)

[Rozdział 3. Inteligentny system wykrywania reklam 13](#_Toc359171434)

[3.1. Wykorzystane narzędzia 15](#_Toc359171435)

[Rozdział 4. Moduł pozyskiwania danych 16](#_Toc359171436)

[4.1. Generowanie adresów URL 16](#_Toc359171437)

[4.2. Wstępna klasyfikacja 17](#_Toc359171438)

[4.3. Implementacja 17](#_Toc359171439)

[Rozdział 5. Moduł przetwarzania wstępnego 19](#_Toc359171440)

[5.1. Ekstrakcja cech 19](#_Toc359171441)

[5.2. Selekcja cech 23](#_Toc359171442)

[5.3. Dodatkowa obróbka 25](#_Toc359171443)

[5.4. Implementacja 26](#_Toc359171444)

[Rozdział 6. Moduł dobóru Klasyfikatora 31](#_Toc359171445)

[6.1. Zakres badań 31](#_Toc359171446)

[6.2. System ewaluacji klasyfikatorów 33](#_Toc359171447)

[6.3. Generator zadań 34](#_Toc359171448)

[6.4. Serwer centralny 36](#_Toc359171449)

[6.5. Węzły klienckie 36](#_Toc359171450)

[6.6. Analizator wyników 37](#_Toc359171451)

[6.7. Implementacja 38](#_Toc359171452)

[Rozdział 7. Moduł trenujący 43](#_Toc359171453)

[7.1. Implementacja 43](#_Toc359171454)

[Rozdział 8. Moduł agent produkcyjnego 44](#_Toc359171455)

[8.1. Implementacja 44](#_Toc359171456)

[Rozdział 9. Wyniki 46](#_Toc359171457)

[9.1. Zebrane dane 46](#_Toc359171458)

[9.2. Skuteczność systemu 47](#_Toc359171459)

[Rozdział 10. Podsumowanie 50](#_Toc359171460)

[10.1. Kierunki rozwoju 50](#_Toc359171461)

[10.1.1. Duże zbiory danych 50](#_Toc359171462)

[10.1.2. Dokładniejsza selekcja cech 51](#_Toc359171463)

[10.1.3. Wstępna klasyfikacja oparta o zachowanie użytkowników 51](#_Toc359171464)

[10.1.4. Klasyfikacja na podstawie kontekstu 51](#_Toc359171465)

[10.1.5. Klasyfikacja na podstawie treści 52](#_Toc359171466)

[10.1.6. Heurystyka wyboru najlepszego klasyfikatora 52](#_Toc359171467)

[Bibliografia 53](#_Toc359171468)

[Spis ilustracji i tabel 54](#_Toc359171469)

[Dodatek A. Zawartość płyty DVD 55](#_Toc359171470)

[Dodatek B. Konfiguracja i uruchamianie systemu 57](#_Toc359171471)

# Wstęp

Wraz z rozwojem Internetu i sieci WWW w bardzo szybkim tempie wzrasta ilość danych przesyłanych łączami telekomunikacyjnymi. Jednak oprócz zwiększania się ilości treści pożądanych przez użytkownika, można zaobserwować także znaczący wzrost liczby reklam. Agencje wykorzystując ciągły przyrost szybkości transferu, umieszczają w sieci coraz obszerniejsze materiały promocyjne: grafiki, animacje, a nawet całe klipy wideo zawierające dźwięk. Im więcej takich elementów zawiera strona internetowa, tym dłużej użytkownik musi czekać na jej załadowanie, a pliki reklam bardzo często mają znaczący udział w ilości danych pobranych przez przeglądarkę klienta. Dodatkowo – w odróżnieniu od pozostałych grafik stanowiących stałe elementy układu graficznego strony – reklamy zwykle nie są zapisywane w pamięci podręcznej przeglądarki. Przy przechodzeniu między kolejnymi stronami portalu efektywnie pobieramy pojedynczy plik HTML i wiele materiałów reklamowych. Nawet małe reklamy w dużych ilościach potrafią spowolnić proces ładowania strony internetowej, ponieważ zmuszają przeglądarkę do wykonania wielu zapytań do wielu różnych serwerów. Sytuacja intensyfikuje się szczególnie w sieciach mobilnych, w których opłaty są naliczane w zależności od objętości transferowanych danych. Użytkownik w tym momencie traci podwójnie, ponieważ nie dość, że zwykle nie ma ochoty oglądać reklam, to jeszcze musi za tą wątpliwą przyjemność płacić.

Z wyżej wymienionych powodów narasta potrzeba oddzielenia rzeczywistej treści stron internetowych od tej niepożądanej, podobnie jak nie tak dawno temu zaczęto oddzielać spam od prawdziwych wiadomości poczty elektronicznej.

## Cel i zakres pracy

Celem pracy jest stworzenie systemu do wykrywania reklam w sieci WWW opartego o metody uczenia maszynowego, który będzie w stanie odróżnić reklamy od treści w sposób nie wymagający ingerencji użytkownika. Istniejące obecnie rozwiązania pozwalają na filtrowanie reklam, jednak są one oparte na czarnych listach serwerów reklamowych, do których wysyłanie żądań HTTP jest blokowane na poziomie przeglądarki internetowej. Mimo swojej dużej skuteczności aplikacje te wymagają częstego i manualnego uaktualniania list adresów. Pojawia się tutaj szerokie pole do zastosowania algorytmów uczenia maszynowego oraz eksploracji danych. Metody te umożliwiają zautomatyzowanie całego procesu, zwiększenie skuteczności filtracji oraz przede wszystkim minimalizację kosztów utrzymania aplikacji.

System powinien być nieinwazyjny i współdziałać ze wszystkimi przeglądarkami internetowymi. Można to osiągnąć poprzez zbudowanie odpowiedniego pośredniczącego (ang. proxy) między przeglądarką, a siecią WWW. Serwer ten może filtrować ruch i odrzucać zapytania HTTP dotyczące materiałów reklamowych. System powinien pozwolić na ocenę i porównanie skuteczności wielu różnych metod uczenia maszynowego takich jak: naiwny klasyfikator Bayesa, sieć Bayesa, klasyfikator k-najbliższych sąsiadów, maszyna wektorów nośnych, sieć neuronowa, drzewo decyzyjne, las drzew losowych, meta-klasyfikator AdaBoost oraz inne. Ponadto musi istnieć możliwość podania różnych zestawów parametrów dla wymienionych metod.

## Powiązane prace naukowe i publikacje

Moja praca magisterska powstała w ramach projektu SmartAdBlocker prowadzonego przez Politechnikę Warszawską na zlecenie T-Mobile. Projekt był prowadzony pod przewodnictwem prof. dr hab. inż. Mieczysława Muraszkiewicza w czteroosobowym zespole:

* Piotr Szczepański (kierownik projektu)
* Adrian Wiśniewski
* Michał Januszewski
* Krzysztof Kamiński

Moim zadaniem było opracowanie architektury i implementacja systemu, co zostało opisane w tej pracy naukowej. Częścią systemu jest serwer pośredniczący, służący do filtrowania żądań HTTP oraz biblioteka blokująca reklamy na podstawie list wyrażeń regularnych. Oba te komponenty zostały opisane w pracy inżynierskiej Michała Januszewskiego [[1](#Mic12)]. Rozległy i dokładny opis charakterystyk wielu stworzonych klasyfikatorów oraz zbudowanego systemu jako całości został wyodrębniony i umieszczony w pracy inżynierskiej Krzysztofa Kamińskiego [[2](#Krz13)]. Ponadto powstała publikacja naukowa opisująca zasady działania systemu napisana przez Piotra Szczepańskiego oraz mnie. Ukazała się w Journal of the Applied Mathematics, Statistics and Informatics [[3](#Szc13)].

# Filtrowanie reklam w sieci WWW

Poruszając się po sieci WWW wysyłamy szereg żądań protokołu HTTP[[1]](#footnote-1). Protokół ten pozwala pobrać ze wskazanego serwera zasób znajdujący się pod podanym adresem URL[[2]](#footnote-2). Najczęściej zasobem tym jest strona internetowa, będąca plikiem tekstowym sformatowanym zgodnie z językiem HTML[[3]](#footnote-3). Plik ten zawiera opis układu strony: definicje różnych elementów, ich wzajemne położenie oraz wskazówki dotyczące ich formatowania. Zasadniczymi elementami pojawiającymi się w tego typu dokumencie są bloki tekstowe zawierające treść, hiperłącza oraz odwołania do innych zasobów takich jak: obrazy, animacje SWF[[4]](#footnote-4), skrypty Javascript i arkusze stylów CSS[[5]](#footnote-5).

Można wyróżnić dwa główne typy reklam. Pierwszym z nich są wyskakujące okna (ang. pop-up). Są to specjalnie przygotowane strony internetowe zawierające jedynie reklamy, które pojawiają się zaraz po wejściu na inną stronę. Jest to najbardziej irytujący typ reklam, ponieważ okno z reklamą zasłania całą treść oraz zmusza użytkownika do przerwania bieżącej czynności i zamknięcia wyskakującego okienka. Początkowo takie reklamy były bardzo popularne, jednak w związku ze zbyt dużą natarczywością reklamodawców w przeglądarkach internetowych wprowadzono mechanizmu blokowania okienek wyskakujących. Spowodowało to znaczny spadek ilości reklam dystrybuowanych tą metodą. Istnieje także odmiana wyświetlająca okienko pod oknem z treścią (ang. pop-under). Działa bardzo podobnie i jest tylko odrobinę mniej irytująca.

Drugim typem są reklamy osadzone umieszczane obok treści jako elementy strony internetowej. Zazwyczaj przybierają formę obrazów lub animacji SWF, ale mogą być to także reklamy tekstowe. Są znacznie mniej inwazyjne, jednak nadal zaciemniają treść, ponieważ znajdują się w najbardziej widocznych i eksponowanych miejscach. Reklamy tego typu występują w wielu kształtach i rozmiarach. Od tapet umieszczanych w tle („watermark”), przez obrazki („button” – 120 na 60 pikseli, „box” – 125 na 125 pikseli, „baner” – 400 na 50 pikseli) do reklam zakrywających prezentowaną treść („toplayer”, „interstitial”). Często są ruchome, co dodatkowo odwraca uwagę użytkownika od czytanej treści. Ponadto niektóre starają się upodobnić do okienek systemu operacyjnego, co jeszcze bardziej dezorientuje i wprowadza w błąd. Dokładny opis różnych rodzajów reklam znajduje się w [[4](#Bra31)].

## Po czym rozpoznać reklamę?

Aby rozpoznać reklamę, musimy określić zbiór cech, które odróżnią ją od treści strony internetowej. Celem budowanego systemu jest minimalizacja danych pobieranych przez sieć, dlatego należy skupić się na cechach, które nie wymagają wysyłania zapytania do serwera.

W przypadku reklam z grupy wyskakujących okienek, jedyną informacją jaką posiadamy jest adres URL żądanej strony. Wyświetlenie tych okienek jest inicjowane najczęściej przez skrypty osadzone w stronie internetowej, co znacząco utrudnia rozpoznanie kontekstu, w jakim dane żądanie zostało zgłoszone. Bez znacznej ingerencji w przeglądarkę nie będziemy posiadać informacji, czy okienko otworzyło się samo, czy w związku z akcją użytkownika np. kliknięciem łącza otwierającego galerię zdjęć.

W przypadku reklam osadzonych znamy dodatkowo kontekst elementu: jego rozmiar, położenie oraz szereg innych atrybutów. Rozmiar jest bardzo obiecującą cechą, ponieważ w przypadku reklam jest zwykle ustandaryzowany – agencja tworzy co najwyżej kilka wersji reklamy i oczekuje, że na wszystkich stronach internetowych, na których reklamy te będą umieszczane, będą one wyglądały dobrze. Natomiast administrator strony internetowej umieszcza reklamy od różnych agencji i nie zmienia przy tym układu swojej witryny. Możemy wykorzystać także informację dotyczącą położenia elementu względem innych elementów strony. Jeżeli mamy do czynienia z dużym podłużnym obrazkiem u góry strony, istnieje duża szansa na to, że jest to baner reklamowy. Podobnie z obrazkami, które przykrywają tekst na stronie. Bardzo często nazwy elementów i klas stylów zdradzają przeznaczenie elementu np.:



Listing : Fragment strony http://www.wp.pl[[6]](#footnote-6) – reklamy znajdują się w blokach div oznaczonych jako adv

Analiza kontekstu elementu na stronie internetowej jest jednak zadaniem bardzo złożonym ze względu na ogromną ilość różnych układów stron[[7]](#footnote-7). Natomiast adresy URL są znacznie krótsze i prostsze, a zawierają wystarczająco informacji aby rozróżnić reklamy od treści. Korzystając z nich odrzucamy na wstępie możliwość filtrowania osadzonych reklam tekstowych, które z racji swojej natury nie są oddzielnym zasobem i nie posiadają adresu URL. Reklamy te pobieramy razem ze stroną internetową, przez co na ich usuwaniu nie zaoszczędzimy transferu. Na poniższym listingu możemy bez trudu zobaczyć jak dużo reklam pobiera przeglądarka, ponieważ angielskie słowo „ads” oznaczające „reklamy” zdradza zawartość zasobów:



Listing : Adresy URL odwiedzone kolejno podczas jednej sesji

Następne cechy możemy uzyskać dopiero po wysłaniu żądania do serwera i odebraniu zasobu. Pozwoli nam to uzyskać informacje o rozmiarze i typie MIME[[8]](#footnote-8) pliku oraz innych nagłówkach HTTP wysłanych przez serwer (w szczególności dotyczących kontroli pamięci podręcznej). Jest to ogromna ilość bardzo użytecznych informacji. Rozmiary większości reklam są ustandaryzowane i nie przekraczają dozwolonej dla danego formatu liczby kilobajtów. Także niewielka liczba typów plików jest używana jako reklamy.

Ponadto da nam dostęp do samej treści zasobu, z której można wydobyć kolejne cechy. W przypadku multimediów można nawet zastosować metody rozpoznawania obrazów i głosu. Taka analiza wymagałaby jednak znaczących nakładów mocy obliczeniowej i obecnie nie jest możliwa ze względów praktycznych, ponieważ powodowałaby olbrzymie opóźnienia w czasie wyświetlania strony.

## Obecne rozwiązania

Istnieje bardzo dużo programów i wtyczek do przeglądarek pozwalających na filtrowanie reklam. Te najprostsze modyfikują plik hosts, który w systemie Windows jest używany do tłumaczenia nazw serwerów na adresy IP. Zmiana polega na przekierowaniu żądań do znanych serwerów reklamowych do niepoprawnego adresu IP, co uniemożliwia pobranie plików z reklamami. Rozwiązanie to wymaga podania całej nazwy hosta i nie adaptuje się w żadnym stopniu do zmieniających się warunków. Pojawienie się nowego serwera reklamowego wymusza konieczność uaktualnienia listy przekierowań, a obejście tego mechanizmu przez agencje jest bardzo proste i wymaga jedynie stworzenia nowej subdomeny, z której będą wysyłane reklamy. Nie jest to trudne i można to robić dowolnie często.

Bardziej złożone rozwiązania – w tym najpopularniejsze takie jak Adblock Plus[[9]](#footnote-9) czy AdBlock[[10]](#footnote-10) – opierają się o listy specjalnych wyrażeń regularnych zwanych filtrami. Adres URL żądania jest porównywany z filtrem i w przypadku dopasowania żądanie jest odrzucane. Niektóre filtry działają jako biała lista, pozwalając na bezpieczne pobranie zasobu. Listy wyrażeń są tworzone i utrzymywane najczęściej niezależnie od samych programów. Najpopularniejszą jest EasyList[[11]](#footnote-11), która jest używana przez ponad 12 milionów użytkowników[[12]](#footnote-12). Zawiera ona reguły blokujące większość dużych i znanych serwerów reklamowych działających w anglojęzycznej części sieci WWW. Ponadto lista ta posiada różne rozszerzenia[[13]](#footnote-13). Większość z nich dodaje reguły filtrujące serwery reklamowe w konkretnych krajach – w tym w Polsce. Istnieją także takie, które blokują dostęp do zasobów związanych ze śledzeniem użytkownika, portalami społecznościami, a nawet takie, które blokują drażniące fragmenty popularnych stron internetowych. Zaletą tego podejścia jest większa możliwość generalizacji. Adresy URL różnych agencji posiadają często podobne fragmenty, co pozwala jednej regule blokować wiele serwerów na raz. Istnieje pewne prawdopodobieństwo, że nowopowstające lub zmieniające się serwery zostaną wyłapane przez istniejące reguły. Niestety ogólność ta czasami skutkuje zablokowaniem poprawnych treści. Najczęściej ofiarą padają blogi poświęcone reklamie w Internecie zawierające w adresie słowa kluczowe, na które reagują filtry. Listy filtrów są publiczne, co z jednej strony pozwala w miarę łatwo je obejść poprzez zmianę formatu adresów, jednak z drugiej sprawia, że są często uaktualniane. Główną wadą jest nadal konieczność ręcznego dodawania reguł przez dużą liczbę osób i związane z tym koszty.

## Metody inteligentne

Skoro istnieją pewne cechy odróżniające reklamy od treści – w tym te dotyczące sposobu budowy adresów URL – do rozwiązania problemu filtrowania reklam w sieci WWW można spróbować wykorzystać algorytmy uczenia maszynowego. Ich przewagą w stosunku do omówionych przed chwilą metod jest to, że są one w stanie wykryć nawet bardzo złożone zależności między cechami pozwalające rozróżnić reklamy od treści. W przypadku przekierowań hostów używamy tylko nazwy domeny. Używając list wyrażeń regularnych dysponujemy adresem URL i pewnymi cechami kontekstu. Wykorzystując algorytmy uczenia maszynowego możemy definiować dowolne atrybuty – związane z nazwą hosta, adresem URL, kontekstem i zawartością zasobu. Ponadto nawet dla samych adresów URL, zastosowanie sztucznej inteligencji pozwala uzyskać lepsze reguły. Osoby tworzące listy filtrów przy dodawaniu nowej reguły polegają na swoim doświadczeniu i intuicji. Komputer mieszczący w pamięci tysiące, a nawet miliony adresów URL, jest w stanie odkryć znacznie więcej subtelnych i nieoczywistych właściwości, niż człowiek. Można liczyć na to, że reguły stworzone w ten sposób z jednej strony będą bardziej ogólne, a z drugiej będą charakteryzowały się mniejszym współczynnikiem błędu, pozwalając na osiągnięcie wyższej skuteczności od obecnych rozwiązań. Istotniejszą kwestią jest redukcja kosztów utrzymania systemu. Metody uczenia maszynowego pozwalają uzyskać reguły zupełnie automatycznie, bez potrzeby zatrudnia zespołu wykwalifikowanych pracowników.

# Inteligentny system wykrywania reklam

Naszym celem jest stworzenie systemu do filtrowania reklam, który będzie wykorzystywał metody uczenia maszynowego. Ograniczamy się do rozpoznawania reklam na podstawie adresów URL. Cały system składa się z pięciu modułów.

Pierwszy zajmuje się pozyskiwaniem danych. Większość algorytmów uczenia maszynowego wymaga istnienia zbioru trenującego, na którym algorytm się uczy. Produktem tego modułu jest lista adresów URL wraz z informacją, czy jest to reklama. Głównym problemem jest dokonanie wstępnej klasyfikacji na bardzo dużych zbiorach adresów. Do tego celu używane są specjalne komponenty tzw. mentorzy.

Gdy już uzyskamy taki spis, drugi moduł musi przetłumaczyć adresy URL na wektory cech, zrozumiałe dla klasyfikatorów. Musimy określić odpowiedni zbiór badanych cech oraz sposób obliczania ich wartości. Dokonujemy w tym momencie także normalizacji i transformacji danych do odpowiednich reprezentacji. Otrzymujemy dwa produkty: dane w postaci macierzy (rzędy odpowiadają poszczególnym adresom, a kolumny cechom) oraz komponent z zapisaną transformatą adresu URL na wektor cech.

Kolejnym i kluczowym etapem jest wybór odpowiedniego klasyfikatora i jego konfiguracji. Używając przygotowanego wcześniej zbioru trenującego, dokonujemy porównania skuteczności różnych algorytmów przy ustawieniu różnych wartości ich parametrów. Mając wyniki przeprowadzonych w ten sposób badań, należy ustalić miarę, na podstawie której zostanie wyłoniony najlepszy klasyfikator.

Czwarty moduł przyjmuje nazwę wyłonionego algorytmu wraz z opisem konfiguracji oraz zbiór danych otrzymany z modułu drugiego. Za pomocą tych składników tworzy klasyfikator docelowy, który zostanie użyty w środowisku produkcyjnym.

Ostatni – piąty – moduł jest agentem będącym produktem systemu. Ocenia on adresy URL, konwertując je do wektora cech za pomocą przygotowanego wcześniej transformatora, a następnie odpytuje wytrenowany klasyfikator. Jest to jedyny moduł, który jest przekazywany użytkownikowi i działa w czasie rzeczywistym. Musi działać niezawodnie i możliwe jak najszybciej. Pozostałe moduły są używane jedynie jako narzędzia i długość ich czasu wykonania, chociaż o całe rzędy wielkości większa, jest niewidoczna dla użytkownika.

**1. Moduł pozyskiwania danych**

* Tworzenie listy adresów URL
* Wstępna klasyfikacja

**2. Moduł przetwarzania wstępnego**

* Ekstrakcja cech
* Selekcja cech
* Dodatkowa obróbka

**3. Moduł doboru klasyfikatorów**

* Dobór klasyfikatora
* Dobór parametrów

**4. Moduł trenujący**

* Stworzenie klasyfikatora

**5. Agent produkcyjny**

* Rozpoznawanie reklam   
  w czasie rzeczywistym

**Wstępny zbiór danych**

{adres URL, czy reklama}

**Zbiór danych**

{cecha1, cecha2, …}

**Transformator**

Adres URL =>

{cecha1, cecha2, …}

**Ustawienia klasyfikatora**

Typ klasyfikatora

Parametry

**Klasyfikator**

{cecha1, cecha2, …}

=> Czy reklama

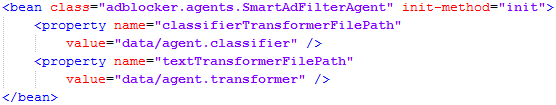
Ilustracja : Schemat inteligentnego systemu rozpoznawania reklam

Oprócz wymienionych modułów w skład systemu wchodzi serwer pośredniczący napisany przez Michała Januszewskiego [[1](#Mic12)]. Jest on używany przez moduł pozyskiwania danych do wygenerowania bazy adresów URL oraz przez moduł agenta, jako docelowe środowisko pozwalające blokować reklamy w przeglądarce internetowej.

## Wykorzystane narzędzia

Podczas budowy systemu postanowiliśmy skorzystać z gotowych i sprawdzonych implementacji algorytmów klasyfikacji i eksploracji danych. Wybór padł na pakiet WEKA[[14]](#footnote-14), który jest bardzo wszechstronnym narzędziem do eksploracji danych, dostępnym na licencji oprogramowania otwartego (ang. Open Source). Oprócz implementacji wielu klasyfikatorów, które można parametryzować na setki sposobów, zawiera także rozwiązania dotyczące filtrowania i transformacji danych, w tym algorytmy selekcji cech, dyskretyzacji oraz wiele innych. Alternatywę stanowiły wyspecjalizowane narzędzia takie jak język R[[15]](#footnote-15) czy Matlab[[16]](#footnote-16), które są często wykorzystywane w kontekście eksploracji danych. Jednak zastosowanie pakietu WEKA pozwala na użycie języka programowania Java, który jest językiem ogólnego przeznaczenia, dzięki czemu kod całego systemu jest jednolity.

Użycie Javy pozwala ponadto na zastosowanie komponentowego paradygmatu programowania [[5](#Szy02)]. Wszystkie kluczowe obiekty wykorzystane w systemie swoją funkcjonalność udostępniają poprzez abstrakcyjne interfejsy. Pozwala to wymienić te obiekty na dowolne inne, pod warunkiem, że będą one także implementować takie same interfejsy. Dzięki temu architektura systemu jest otwarta i bardzo rozszerzalna. Ponadto do zarządzania komponentami wykorzystujemy bibliotekę Spring IOC[[17]](#footnote-17), która pozwala w pliku XML określić klasy, właściwości oraz powiązania między wszystkimi komponentami. Pozwala to uzyskać kontrolę nad każdym, nawet najdrobniejszym elementem systemu z poziomu edytora tekstu.



Ilustracja :Przykład konfiguracji komponentu agenta w Spring IOC

# Moduł pozyskiwania danych

## Generowanie adresów URL

Sieć WWW jest olbrzymim źródłem danych. Na chwilę obecną szacuje się, że indeks wyszukiwarki internetowej Google obejmuje ponad czterdzieści pięć miliardów różnych stron internetowych[[18]](#footnote-18). Przetworzenie tak dużej ilości informacji jest poza zasięgiem naszych możliwości obliczeniowych, mimo tego, że dysponujemy maszynami posiadającymi nawet kilkaset rdzeni. Z tego względu testy systemu zostaną przeprowadzone jedynie na zbiorze popularnych polskich stron internetowych.

Listę stron tego typu możemy wygenerować na dwa sposoby. Pierwszy z nich to uruchomienie robota internetowego, który przeszukuje sieć WWW w ten sposób, że zaczynając od zbioru podanych adresów URL, pobiera kolejne strony internetowe, odnajduje zamieszczone na nich odnośniki, a następnie rekursywnie pobiera powiązane z nimi zasoby. Proces powtarza możliwie długo (zwykle do przerwania przez użytkownika), przechodząc między wieloma stronami internetowymi. Pozwala to uzyskać bardzo dużo adresów, gęsto pokrywających sieć w krótkim czasie.

Drugim sposobem jest śledzenie ruchu prawdziwych użytkowników. Można to osiągać za pomocą serwera pośredniczącego, który w tle zapisuje wszystkie żądania użytkownika. Mimo, że sposób ten jest znacznie bardziej kosztowny zarówno jeżeli chodzi o czas jak i nakład pracy, dane zebrane w ten sposób są znacznie lepsze jakościowo. Lista adresów uzyskana w ten sposób odpowiada rozkładowi z jakim system będzie odpytywany w warunkach produkcyjnych, co pozwala wszystkim klasyfikatorom na uzyskanie lepszej skuteczności. Każdy klasyfikator opisuje pewną n-wymiarową przestrzeń cech, w której kolejne adresy są pojedynczymi punktami. Jeżeli klasyfikator będzie odpytywany w okolicy punktów, na których został wytrenowany, możemy oczekiwać lepszych wyników niż wtedy, gdy będziemy prosili o ocenę obszarów słabo przez niego zbadanych. Najlepiej widać to na przykładzie klasyfikatora k-najbliższych sąsiadów, którego decyzja zależy od tego, do jakiej klasy należało k punktów ze zbioru trenującego leżących najbliżej punktu o który pytamy. Innym przykładem jest naiwny klasyfikator Bayesa, który estymuje prawdopodobieństwa a priori tego, czy zasób jest reklamą, czy nie. Wartości te mogą skrajnie się różnić w przypadku syntetycznie wygenerowanego ruchu, który odwiedza wszystkie strony dokładnie raz i przegląda dokładnie wszystkie zasoby napotkanych portali. Zwykli użytkownicy przeglądarek internetowych wielokrotnie częściej odwiedzają strony główne portali, na których jest zazwyczaj dużo reklam, a wielu podstron nie wyświetlają nigdy. Decyzją projektową postanowiliśmy skorzystać z drugiego sposobu i przez kilka dni w czteroosobowym zespole zebraliśmy 55 202 adresów URL.

## Wstępna klasyfikacja

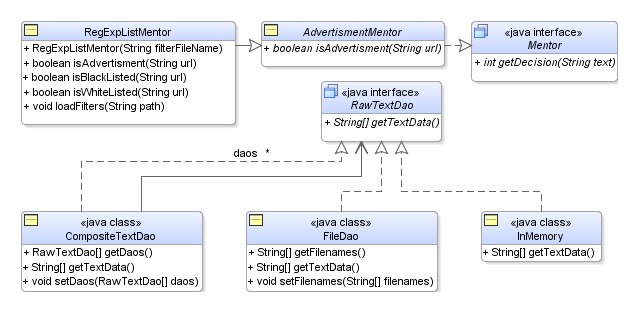
Aby nauczyć klasyfikatory prawidłowo rozpoznawać reklamy, musimy przy każdym zebranym adresie podać informację czy jest on reklamą, czy treścią. Można osiągnąć to na trzy sposoby. Najbardziej oczywistym jest ręczna klasyfikacja zebranych adresów, jednak przy dużej ilości adresów jest to zbyt pracochłonne i zupełnie nieopłacalne rozwiązanie. Drugi sposób polega na wykorzystaniu istniejących rozwiązań do filtrowania reklam, na przykład listy EasyList. Dzięki temu w ciągu kilku sekund możemy sklasyfikować dowolny zbiór adresów URL. Należy jednak pamiętać, że postępując w ten sposób, budowany klasyfikator jest obarczony dodatkowym błędem wynikającym z błędów zawartych w wykorzystywanym rozwiązaniu. Na szczęście możemy liczyć, że przynajmniej część tej niedokładności zostanie wyeliminowana przez właściwość generalizacji stosowanych algorytmów.

Trzeci sposób polega na umożliwieniu użytkownikowi blokowania i odblokowywania poszczególnych zasobów. Mechanizm ten i tak musi być wbudowany w system, ponieważ nawet najdokładniejszy klasyfikator może się pomylić. Ważne jest, aby użytkownik mógł odblokować treść, która została odrzucona jako reklama. Mając informację o żądaniach blokowanych i odblokowywanych możemy wnioskować o ich klasie. Rozwiązanie to także uwidacznia i pozwala szybko reagować na błędy systemu. Jeżeli użytkownicy często ręcznie kontrolują zasób, można go dodać do zbioru trenującego i zaktualizować klasyfikator. Może się to dziać w pełni automatycznie. Metoda ta niestety nie jest odpowiednia przy rozruchu systemu ze względu na czas potrzebny do osiągnięcia zadowalającej skuteczności klasyfikacji.

Do wstępnej klasyfikacji postanowiliśmy wykorzystać listę filtrów EasyList wraz z polskim suplementem. W ten sposób natychmiast uzyskaliśmy dosyć dokładną klasyfikację dla całego uprzednio zebranego zbioru adresów.

## Implementacja

Moduł pozyskiwania danych nie posiada własnego pliku wykonywalnego i składa się z zestawu komponentów wykorzystywanych przez inne moduły. Adresy URL są wczytywane do systemu poprzez interfejs RawTextDao. Posiada on jedną metodę, która zwraca tablicę adresów, stanowiących zbiór danych. Podstawową implementacją jest FileDao, które czyta adresy z pliku tekstowego sformatowanego w ten sposób, że w każdej linii znajduje się dokładnie jeden adres. CompositeTextDao jest kompozytem umożliwiającym wczytywanie danych z wielu obiektów implementujących RawTextDao, w naszym przypadku z wielu różnych plików. Implementacja InMemory jest makietą testową (ang. Mock Object), która zwraca tablicę na stałe zapisaną w kodzie. Pliki z adresami URL są generowane przez serwer pośredniczący, który zapisuje każdy pobrany zasób.



Ilustracja : Kluczowe klasy modułu pozyskiwania danych

Wstępna klasyfikacja jest natomiast wykonywana przez interfejs Mentor, którego zadaniem jest określenie klasy dla podawanego adresy URL. Klasa abstrakcyjna AdvertisementMentor adaptuje metodę getDecision, w ten sposób, że w odpowiedzi na zawołanie isAdvertisement zwraca odpowiedni numer klasy (0 dla treści i 1 dla reklam). Implementacja RegExpListMentor do określenia czy zasób jest reklamą wykorzystuje listy wyrażeń regularnych. Szczegóły implementacji interpretera wyrażeń można znaleźć w pracy inżynierskiej Michała Januszewskiego [[1](#Mic12)]. Poprzez implementację interfejsu Mentor można dodać dowolne inne metody wstępnej klasyfikacji.

# Moduł przetwarzania wstępnego

## Ekstrakcja cech

Adresy URL w postaci ciągów znaków nie są zrozumiałe dla algorytmów klasyfikacji. Oczekują one opisania wszystkich próbek za pomocą wektorów wartości numerycznych. Każda z tych wartości jest odzwierciedleniem pewnej ustalonej cechy adresu. Zbiór wszystkich cech tworzy n-wymiarową przestrzeń w której umieszczane są poszczególne adresy. Żeby można było odróżnić treść od reklam, przestrzeń ta musi być na tyle duża, aby punkty odpowiadające różnym klasom nie pokrywały się ze sobą, a najlepiej tworzyły oddalone od siebie skupiska.

Zagadnieniem ekstrakcji cech z ciągów znaków zajmuje się dziedzina eksploracji tekstu[[19]](#footnote-19). Podstawowym ich źródłem jest podział tekstu na symbole (ang. token) – fragmenty tekstu niosące pewną informację. Najczęściej są to pojedyncze słowa, których ilość jest następnie zliczana i umieszczana w wektorze cech jako częstość występowania danego symbolu. Adresy URL reklam znacznie częściej zawierają słowa związane z tą branżą, w szczególności różne formy słowa ad (z angielskiego: reklama) oraz liczby oznaczające typowe wymiary banerów. Klasyfikatory są w stanie zapamiętać także nazwy agencji reklamowych, które bardzo często występują w nazwie hosta.

Sam podział tekstu na pojedyncze słowa nie jest jednak czynnością trywialną. Typowe heurystyki łamią ciągłe fragmenty tekstu w miejscu, gdzie znajdują się spacje lub znaki interpunkcyjne. Nasze rozwiązanie także korzysta z takiej metody. Adresy URL są jednak dosyć specyficzne i różnią się od zwykłego tekstu. Z jednej strony są znacznie prostsze, ponieważ zwykle nie zawierają znaków interpunkcyjnych (średnik zdarza się rzadko), ale z drugiej nie mogą zawierać spacji[[20]](#footnote-20), przez co bardzo często mamy do czynienia ze zbitkami słownymi typu: googleads, pagead itp. Problem ten można rozwiązać wykorzystując słownik i metodę redukcji entropii próbując łamać napotkane symbole na mniejsze[[21]](#footnote-21), jednak nie zdecydowaliśmy się na takie rozwiązanie, ponieważ dużo takich związków jest nazwami charakterystycznymi dla serwerów reklamowych i nie występuje nigdzie indziej np. w adresach do blogów reklamowych często zawierających słowo ad. Kolejnym utrudnieniem są słowa występujące w różnych formach i przypadkach. Istnieje wiele gotowych bibliotek, które poprzez ekstrakcję rdzenia słowa (ang. stemming) rozwiązują ten problem, jednak są one zorientowane na konkretne języki i nie radzą sobie dobrze z tekstami mieszanymi. Na szczęście problem ten praktycznie nie występuje ze względu na specyficzne zasady konstrukcji adresów URL.

Oprócz częstotliwości występowania danych symboli, interesującą informację stanowi także kolejność ich występowania. Dysponując zbiorem wyrazów tworzących pojedynczy adres URL, możemy zbudować krotki – zwane n-gramami – łącząc sąsiadujące ze sobą wyrazy parami lub trójkami. Można utworzyć także luźne n-gramy, które oprócz bezpośrednich sąsiadów dopuszczają wszystkie kombinacje, w których zachowana jest kolejność następowania symboli. Ze względu na bardzo dużą ilość tworzonych w ten sposób cech, nie stosujemy n-gramów dłuższych niż bigramy i trigramy.

Oprócz standardowych metod eksploracji tekstu, przeznaczonych dla dowolnego rodzaju tekstu, możemy wykorzystać także informacje zawarte w samej strukturze adresu URL. Adres ten składa się z kilku różnych części[[22]](#footnote-22) między innymi: nazwy hosta, numeru portu, protokołu, ścieżki i zapytania. Znaczenie symboli może być inne w zależności od miejsca ich wystąpienia. Jeżeli klasyfikator nauczy się odrzucać żądania kierowane do pewnej domeny, to może popełnić błąd w przypadku, gdy jej nazwa wystąpi w ścieżce zapytania (na przykład jako nazwa wpisu zamieszczonego na blogu), a nie części przeznaczonej na nazwę hosta. Informację tę dodajemy poprzez doklejenie do każdego symbolu prefiksu opisującego fragment, z którego on pochodzi np.: Host$, Path$, Query$. Znak dolara występuje tutaj w roli separatora – żaden symbol nie może go zawierać, dzięki czemu cechy stworzone w ten sposób nie mieszają się ze stworzonymi wcześniej.

Kolejną interesującą kwestią jest to, że w odnośnikach występuje ogromna ilość różnych słów. Znajdują się w nich nazwy własne, nazwy parametrów (często skróty i niezrozumiałe ciągi znaków) oraz wartości tych parametrów (nierzadko będące liczbami lub nawet zakodowanymi ciągami BASE64[[23]](#footnote-23)). Na stworzonym przez nas zbiorze trenującym ich liczba wyniosła 33 595, co bezpośrednio przekłada się na liczbę cech. Ponadto z symboli tych są tworzone n-gramy, co dodatkowo powoduje eksplozję kombinatoryczną. Łącznie uzyskaliśmy 330 542 cech. Adresy URL składają się jednak z co najwyżej kilku, kilkunastu słów. Z tego względu większość cech dla danego adresu ma wartość równą zero. Ważne jest, aby macierz cech zapisywać w postaci rzadkiej, ponieważ umożliwia to zmieszczenie zbioru trenującego w pamięci i znacząco przyspiesza jego przetwarzanie.

Poza cechami opartymi na liczeniu częstości występowania symboli, występują także inne interesujące własności odnośników. Długość adresu URL wyrażona w ilości znaków jest zwykle trochę większa w przypadku reklam, niż w zwykłych stron internetowych. Informacja o tym, czy konkretny komponent adresu istnieje, jaka jest jego długość oraz ile symboli zawiera też okazuje się przydatna. W szczególności, jeżeli obecny jest numer portu lub informacje dotyczące uwierzytelniania, możemy być prawie pewni, że nie mamy do czynienia z reklamą. Podobnie, jeżeli żądanie nie zawiera parametrów (w przypadku reklam są one używane do śledzenia skąd pochodzi wyświetlenie). Inną ciekawą cechą – napotkaną w artykule [[6](#Min)] – jest ilość symboli numerycznych występujących w adresie. Poza charakterystycznymi rozmiarami banerów, liczby zawarte w odnośniku niosą mało informacji, bo zwykle funkcjonują jako identyfikatory. Można jednak zauważyć, że w przypadku reklam występuje ich zwykle kilka. Prześledźmy proces ekstrakcji cech na przykładzie dwóch podobnych adresów:

Tabela : Cechy na podstawie przykładowych adresów

|  |  |  |
| --- | --- | --- |
| **Adres URL** | http://whois.domaintools.com/ pagead2.googlesyndication.com | http://pagead2.googlesyndication.com/simgad/6011371405013281332 |
| **Czy reklama?** | NIE | TAK |
| **Surowe symbol** | http – 1  whois – 1  domaintools – 1  com – 2  pagead2 – 1  googlesyndication – 1 | http – 1  pagead2 – 1  googlesyndication – 1  com – 1  simgad – 1  6011371405013281332 – 1 |
| **Symbole z informacją o komponencie URL** | Protocol$http – 1  Host$whois – 1  Host$domaintools – 1  Host$com – 1  Path$pagead2 – 1  Path$googlesyndication – 1  Path$com – 1 | Protocol$http – 1  Host$pagead2 – 1  Host$googlesyndication – 1  Host$com – 1  Path$simgad – 1  Path$6011371405013281332 – 1 |
| **Bigramy sekwencyjne[[24]](#footnote-24)** | http>com – 1  com>domaintools – 1  domaintools>whois – 1  whois>pagead2 – 1  pagead2>googlesyndication – 1  googlesydication>com – 1 | http>com – 1  com>googlesyndication – 1  googlesyndication>pagead2 – 1  pagead2>simgad – 1  simgad>6011371405013281332 – 1 |
| **Bigramy luźne** | http>>com – 2  http>>domaintools – 1  http>>whois – 1  http>>pagead2 – 1  http>>googlesyndication – 1  … (pozostałe 16) | http>>com – 1  http>> googlesyndication – 1  http>> pagead2 – 1  http>> simgad – 1  http>> 6011371405013281332 – 1  … (pozostałe 10) |
| **Długość adresu** | 59 | 63 |
| **Długość komponentów adresu** | Protocol: 4  Host: 21  Path: 29 | Protocol: 4  Host: 29  Path: 26 |
| **Brakujące komponenty adresu** | Port  Auth  Query | Port  Auth  Query |
| **Ilość symboli względem komponentów adresu** | Protocol: 1  Host: 3  Path: 3 | Protocol: 1  Host: 3  Path: 2 |
| **Ilość symboli numerycznych** | 0 | 1 |

Podany odnośnik do reklamy nie zawiera wielu cech charakterystycznych. Nie jest szczególnie długi (rekordowe odnośniki mają nawet 1187 znaków[[25]](#footnote-25)), ani nie zawiera wielu atrybutów numerycznych, ale charakterystyczne słowo ad występuje zarówno w zbitce pagead2, jak i sigmad. Pierwszy z tych symboli pojawia się także w normalnych linkach, co sprawia, że jest mało użyteczny, chyba że weźmiemy pod uwagę informację o komponencie, z którego pochodzi (liczbę wystąpień Host$pagead2). Występujący identyfikator numeryczny (symbol 6011371405013281332) raczej nie będzie się powtarzał w innych adresach, przez co informacja jaką niesie częstotliwość jego występowania jest dla nas nieprzydatna. Na tak małym zbiorze danych, klasyfikator mógłby skorzystać z wielu cech, które pozwalają odróżnić oba adresy, jednak wraz z zwiększaniem liczby próbek w zbiorze trenującym, cechy określające reklamy stają się znacznie bardziej wyraźne.

## Selekcja cech

Podczas ekstrakcji powstaje wiele cech. Część z nich ma istotne znacznie na przykład: pojawienie się nietypowych komponentów adresu URL , symbole związane z nazwami serwisów reklamowych, czy charakterystycznymi dla reklam parametrami. Inne z kolei niosą mniej informacji lub nawet nie zawierają jej wcale. Są to symbole z unikalnymi numerycznymi identyfikatorami, symbole pojawiające się bardzo często (np.: com, pl) albo większość n-gramów ze względu na ich małą powtarzalność. Ponadto niektóre cechy są ze sobą mocno skorelowane. Jest to spowodowane istnieniem grup symboli, które zawsze występują razem. Najczęściej zjawisko to zachodzi wśród nazw parametrów żądania. Cechy redundantne nie wnoszą dodatkowych informacji przez co są nieprzydatne podczas klasyfikacji.

Na zebranym przez nas zbiorze danych, udało się stworzyć dokładnie 330 542 cechy. Z tak dużą liczbą wiążą się pewne problemy. Cechy niosące znikomą ilość informacji mogą być postrzegane jako szum i powodować spadek skuteczności klasyfikacji. Niektóre klasyfikatory są szczególnie wrażliwe na dane tego typu. Na przykład klasyfikator k-najbliższych sąsiadów odpytany o adres mający kluczowe cechy charakterystyczne dla reklamy, ale wiele cech nadmiarowych o losowych wartościach, może uznać, że adres ten leży w dużej odległości od grupy reklam. Natomiast łącze mające kluczowe cechy charakterystyczne dla treści, a cechy nadmiarowe podobne do reklam, na których trenowano klasyfikator, może być mylnie rozpoznany pozytywnie. Dzieje się tak, ponieważ cech nadmiarowych jest znacznie więcej, przez co mają większy wkład w obliczanie odległości między próbkami. Drugim – i znacznie poważniejszym – problemem jest wydłużenie czasu potrzebnego na wytrenowanie klasyfikatora. Dla takiej ilości cech zadanie to nie jest wykonywalne w rozsądnym czasie. Najdłuższa próba użycia całego zbioru do budowy drzewa decyzyjnego trwała ponad trzy doby i została przerwana. Redukcja ilości cech poprzez selekcję najbardziej istotnych jest nieuchronna.

Pierwszym krokiem jest odrzucenie cech, które występują bardzo rzadko. Są one szczególnie niekorzystne, ponieważ powodują nadmierne dopasowanie się klasyfikatora do danych (ang. overfitting). Jeżeli cecha występuje tylko w jednej próbce (we wszystkich pozostałych ma wartość równą zero), może ona zostać wykorzystana do stworzenia reguły adresującej ten jeden konkretny przykład. Reguła ta będzie poprawna na zbiorze testowym, co zwiększy skuteczność klasyfikatora. Nie mamy jednak żadnej informacji o tym w jakich innych kontekstach cecha ta może się pojawić, przez co podczas działania w warunkach produkcyjnych, klasyfikator będzie na ślepo traktował wszystkie odnośniki z tą cechą za pomocą wyjątkowej reguły. W naszym przypadku odrzucamy wszystkie atrybuty występujące w mniej niż pięciu próbkach. Dzięki temu zmniejszamy zbiór cech do 65 154 elementów (265 388 odrzucono). Odrzucone cechy to w większości symbole zawierające rzadkie identyfikatory numeryczne, dane binarne oraz rzadkie n-gramy.

Kolejną metodą jest wykorzystanie specjalnych algorytmów selekcji cech (ang. feature selection), które próbują estymować przydatność cech. Pozwala to wyeliminować niepotrzebne i zachować jedynie użyteczne wartości. Algorytmy te dzielą się na dwie zasadnicze grupy: oceniające poszczególne cechy niezależnie oraz oceniające zbiory cech. Pierwsze nie biorą pod uwagę korelacji między cechami, co skutkuje pozostawieniem wielu cech redundantnych oraz usunięciem takich, które dopiero złożone z innymi okazują się ważne. Drugie nie mają tej niedoskonałości, jednak przestrzeń wszystkich możliwych podzbiorów jest znacznie większa (2n), przez co algorytmy te mają odpowiednio większą złożoność obliczeniową. Ostatecznie najczęściej korzystają z heurystyk zachłannych do budowy najlepszego zbioru, konstruując go element po elemencie, co także nie gwarantuje optymalnego rozwiązania.

Nasze rozwiązanie stosuje metodę hybrydową. Najpierw wykorzystujemy heurystykę przyrostu informacji, która oblicza przydatność cechy jako redukcję entropii:

Równanie : Przyrost informacji cechy f na zbiorze danych X

Wartość progowa przyrostu informacji jest ustawiona w takich sposób, aby zachowywać tylko 5000 najlepszych cech. Dzięki temu otrzymujemy znacznie zmniejszony zbiór cech niosących pewną wymierną ilość informacji. Jednakże dalej pozostają w nim cechy redundantne, dlatego w drugim przebiegu używamy algorytmu CFS (ang. Correlation feature selection), który w każdym kroku do budowanego zbioru zachłannie dodaje element według funkcji:

Równanie : Funkcja oceny przydatności algorytmu CFS

Funkcja ta dla zbioru zawierającego cech rośnie razem ze średnią współczynników korelacji między cechami a klasą i maleje razem ze średnią współczynników korelacji między cechami zbioru . Pozwala to odfiltrować cechy redundantne. Limit wielkości końcowego zbioru cech ustalono poprzez liczne testy na 2000 elementów, co stanowi jedynie 0,6% początkowego rozmiaru.

## Dodatkowa obróbka

Na przygotowanym zbiorze danych są wykonywane kolejne operacje. Pierwszą z nich jest transformacja sposobu reprezentacji cech związanych z symbolami. W fazie ekstrakcji powstają cechy liczące ilość wystąpień poszczególnych symboli w odnośnikach. Wartość tych cech jest zapamiętywana jako liczba naturalna (łącznie z zerem). Jednak w adresach URL powtórzenia symboli zdarzają się bardzo rzadko i informacja ta może okazać się nadmiarowa. Aby do tego nie dopuścić możemy pamiętać jedynie, czy dany symbol wystąpił, czy też nie.

Niektóre symbole mają większe znaczenie niż inne, co nie jest odzwierciedlone ani w postaci częstotliwościowej, ani binarnej. Częstotliwość wystąpienia symboli możemy ważyć na przykład zgodnie z metodą TFIDF[[26]](#footnote-26). Wartość cechy wyliczamy wtedy zgodnie ze wzorem:

Równanie : Ważona częstotliwość występowania symbolu t w dokumencie d ze zbioru dokumentów D

Reprezentacja ta przypisuje dużą wartość, gdy symbol występuje wiele razy w niewielkim zbiorze dokumentów, mniejszą jeżeli symbol występuje w wielu dokumentach i bliską zeru, jeżeli symbol występuje w prawie wszystkich dokumentach. Pozwala to uprzywilejować cechy charakterystyczne dla pewnych grup dokumentów i zmniejszyć znaczenie cech pospolitych.

Niektóre klasyfikatory wymagają stosowania jedynie dyskretnych cech nominalnych (wyliczeń) o znanej z góry i ograniczonej liczbie możliwych wartości. Natomiast większość prezentowanych do tej pory cech jest liczbami rzeczywistymi bez żadnych ograniczeń dotyczących ich maksymalnej wartości. Do klasyfikatorów takich należą między innymi naiwny klasyfikator bayesowski i sieć bayesowska, które liczą prawdopodobieństwo wystąpienia danej wartości cechy w obu rozpatrywanych klasach. Jeżeli cechy mają wartości ciągłe to prawdopodobieństwo często ma wartość równą zero, ponieważ w zbiorze trenującym nie istnieje przykład o dokładnie takiej samej wartości cechy. W konsekwencji uniemożliwia to podjęcie decyzji. Stąd istnieje potrzeba redukcji dziedziny stworzonych cech do ograniczonej liczby wartości. Transformację tą nazywamy dyskretyzacją i może ona być wykonana na wiele sposobów. W naszym projekcie używamy metody MDL [[7](#Fay93)], która automatycznie dobiera zakresy dyskretyzowanych wartości, biorąc pod uwagę klasę.

Ostatnią czynnością wykonywaną na zbiorze danych jest grupowanie zgromadzonych próbek. Istnieje duża liczba adresów, które w zebranym zbiorze danych się powtarzają. Ponadto podczas etapu selekcji cech, różnice między próbkami dodatkowo się zacierają, ponieważ większość cech zostaje odrzucona. Powstaje sytuacja w której wielu różnym adresom URL może zostać przyporządkowany identyczny wektor cech. W celu przyspieszenie dalszych obliczeń, do każdej próbki dodajemy dodatkowo jej wagę, która wyraża bezpośrednio liczbę wystąpień próbki w zbiorze danych. Następnie przeglądamy cały zbiór danych w poszukiwaniu zduplikowanych wektorów cech, każdą taką grupę zapisujemy za pomocą jednego wektora cech z wagą równą liczności grupy. Pozwala to zredukować liczbę przetwarzanych przykładów z 55 202 do 12 411 przy 2000 cechach ciągłych lub nawet 3784 przykładów przy 100 cechach dyskretnych. Okazuje się, że są to najczęściej odnośniki do prawdziwej treści, których większość cech przyjmuje wartość równą zero. System umożliwia także skorzystanie z szeregu transformacji udostępnionych przez wykorzystywany pakiet eksploracji danych.

Efektem końcowym całego modułu przetwarzania wstępnego jest powstanie pliku tekstowego zawierającego opis wszystkich zebranych adresów URL w postaci wektorów w ustalonej przestrzeni cech. Oprócz tego powstaje także obiekt transformatora, potrafiący tłumaczyć pojedyncze adresy URL na wektory cech, wykonując wszystkie opisane w tym rozdziale operacje. Obiekt ten jest zapisywany do pliku binarnego, w celu późniejszego użycia przez moduł agenta.

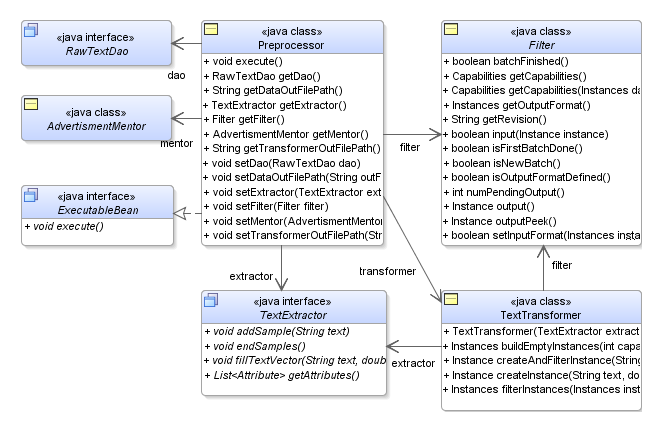
## Implementacja

Każda ze zbudowanych w ramach projektu aplikacji w metodzie Main wywołuje statyczną metodę pomocniczą executeApplication z klasy adblocker.utils.Spring. Metoda ta tworzy kontener komponentów, wczytuje wskazany plik konfiguracyjny, inicjalizuje wszystkie komponenty, po czym szuka wśród nich komponentu implementującego interfejs ExecutableBean i wykonuje jego metodę execute. Efektywnie klasy implementujące ten interfejs stanowią punkt startowy aplikacji działających wewnątrz kontenera komponentów. Wewnątrz modułu przetwarzania jest to klasa Preprocess, której zadaniem jest koordynacja wykonania wszystkich etapów przetwarzania wstępnego. W pierwszej kolejności za pomocą interfejsu RawTextDao wczytywany jest zbiór adresów. Następnie przy pomocy interfejsu Mentor tworzona jest pierwsza cecha adresu – klasa, której będą się uczyć klasyfikatory. Trzecim etapem jest stworzenie reszty cech poprzez ich ekstrakcję z tekstu adresów, służy do tego interfejs TextExtractor. Jako że większość implementacji tego interfejsu wymaga podzielenia adresu URL na symbole, stworzono klasę CompositeUrlExtractor, która po wykonaniu tej czynności przekazuje sterowanie do zbioru komponentów implementujących interfejs UrlSegmentsExtractor operujących na listach symboli. W ten sposób procedura dzielenia wykonywana jest tylko raz. Klasa ta jest połączeniem wzorców adaptera (zmiana reprezentacji z czystego tekstu na listy symboli) oraz kompozytu (przekazanie wykonania do wielu obiektów typu UrlSegmentsExtractor). Istnieją następujące implementacje interfejsu UrlSegmentsExtractor:

* SimpleTokenExtractor – liczy wystąpienia każdego symbolu i dodaje je jako cechy. Podczas zliczania może uwzględnić to z którego komponentu adresu symbol pochodził.
* N\_gramExtractor – liczy wystąpienia n-gramów. Można podać czy interesują nas zbitki tworzone z wyrazów sąsiadujących bezpośrednio, czy też nie. Wartość n także można zmienić.
* UrlNumericCountExtractor – liczy liczbę wystąpień symboli numerycznych w całym adresie oraz jego komponentach.
* SegmentsMissingExtractor – informuje czy komponent adresu występował.
* SegmentLengthExtractor – liczy ilość znaków symboli adresu oraz poszczególnych komponentów.
* TokenCountExtractor – liczy ilość symboli w adresie i jego komponentach.

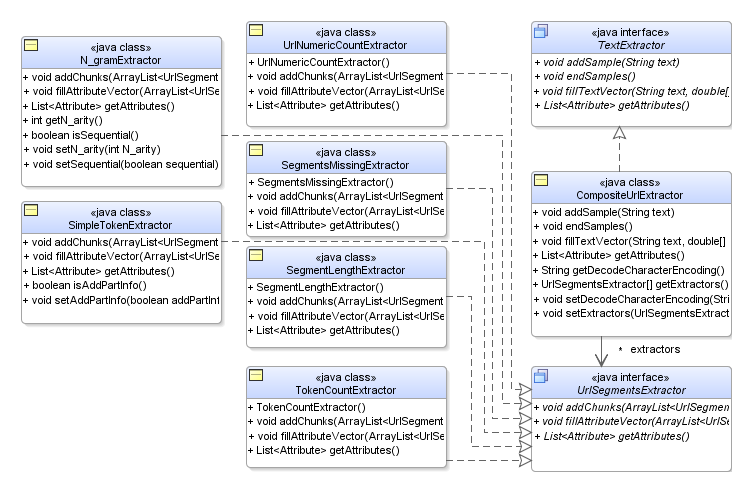
Stworzony w ten sposób zbiór danych jest macierzą liczb zmiennoprzecinkowych o wymiarach równych liczbie adresów URL i liczbie wydobytych cech. Macierz ta jest zapisywana jako obiekt Instances z pakietu WEKA. Zawiera on tablicę obiektów typu SparseInstance, które reprezentują poszczególne wektory cech w postaci rzadkiej przy użyciu tablicy mieszającej, co pozwala zaoszczędzić bardzo dużo miejsca. Przy 55 202 próbkach i 330 542 cechach ilość pamięci potrzebna do przechowania surowej macierzy liczb wynosi ponad 135GB.

Następną czynnością wykonywaną na zbiorze jest selekcja cech oraz wykonanie różnych dodatkowych transformacji. Pakiet WEKA udostępnia klasę abstrakcyjną Filter, która pozwala przekształcać obiekt Instances. Implementacja MultiFilter pozwala w szczególności na wykonanie szeregu różnych przekształceń za pomocą wielu filtrów uruchamianych kolejno. Pierwszym uruchamiany przez nas filtrem jest napisany przez nas FrequentAttribute, który odrzuca atrybuty występujące w zbyt małej lub zbyt dużej liczbie próbek. Wartości te można podać zarówno procentowo jak i bezwzględnie. Następnie filtr AttributeSelection (z pakietu weka.filters) wybiera przydatne atrybuty za pomocą heurystyki AttributeSelection (z pakietu weka.attributeSelection). Istnieje wiele heurystyk, my wykorzystujemy InfoGainAttributeEval oraz CfsSubsetEval. Ostatnim wykorzystywanym filtrem jest MergeSameInstances, który skleja próbki o takim samym zestawie cech, odpowiednio zwiększając ich wagę. Opcjonalnie można przed nim wykonać pewne dodatkowe transformacje np. ToBinaryVector, Discretize lub dowolne inne.

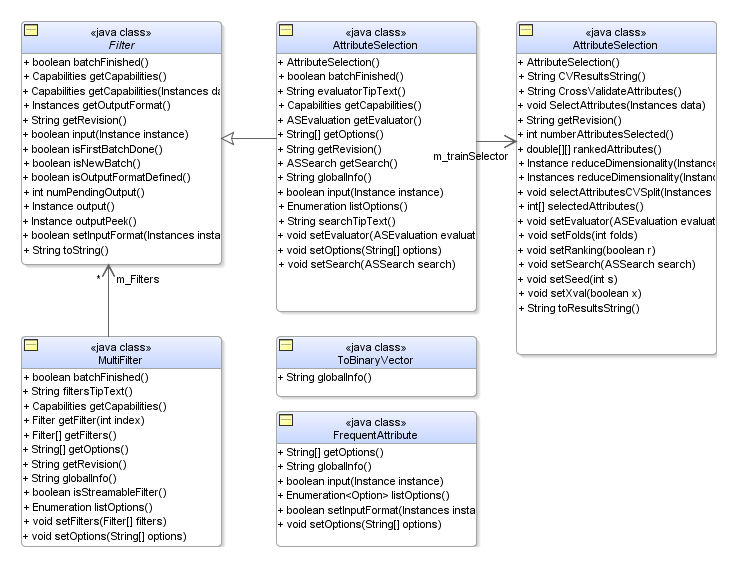


Ilustracja : Kluczowe klasy modułu przetwarzania wstępnego

Gdy zbiór danych jest już gotowy, zostaje zapisywany do pliku tekstowego zgodnie z formatowaniem arff[[27]](#footnote-27). Oprócz zbioru danych zapisywany jest także obiekt TextTransformer, który potrafi na bieżąco konwertować adresy URL do próbek pakietu WEKA. Wykorzystuje on w tym celu dokładnie te same ekstraktory cech oraz filtry.



Ilustracja : Klasy ekstraktorów



Ilustracja : Klasy filtrów (transformacji) danych

# Moduł dobóru Klasyfikatora

Mając dane w ostatecznej postaci możemy przystąpić do trenowania klasyfikatora. Najpierw musimy jednak zdecydować, który z algorytmów klasyfikacji użyć. Jest ich bardzo dużo, a każdy posiada własną charakterystykę i zestaw parametrów, których modyfikacja potrafi diametralnie zmienić ostateczną skuteczność. Ponadto nie można przewidzieć jak zachowają się na zbudowanym zbiorze danych. Jedynym skutecznym sposobem na dobór optymalnego rozwiązania jest metoda prób i błędów. Zadaniem tym zajmuje się moduł ewaluacji klasyfikatorów, który wykonując serię eksperymentów określa rodzaj i konfigurację najlepszego algorytmu.

## Zakres badań

Ze zbioru wszystkich dostępnych klasyfikatorów postanowiliśmy zbadać charakterystykę następujących algorytmów wraz z wymienionymi parametrami:

* Naiwny klasyfikator bayesowski:
  + rodzaje dyskretyzacji;
* Sieć bayesowska:
  + heurystyki budowy sieci,
  + metody oceny sieci,
  + estymatory prawdopodobieństwa;
* Las losowy:
  + liczba drzew,
  + liczba losowych cech użytych do budowy drzewa,
  + maksymalna wysokość drzewa,
  + ziarno generatora liczb losowych;
* Drzewo decyzyjne:
  + metody przycinania (ang. pruning),
  + minimalna liczba próbek w liściach,
  + podziały binarne/wielowartościowe,
  + wygładzanie Laplace'a obliczonych prawdopodobieństw,
  + ziarno generatora liczb losowych;
* AdaBoost:
  + liczba iteracji,
  + procent wag używanych do trenowania klasyfikatora,
  + przepróbkowywanie (ang. resampling),
  + ziarno generatora liczb losowych;
* Sieć neuronowa:
  + stała uczenia,
  + bezwładność propagacji wstecznej,
  + liczba epok,
  + liczba warstw,
  + liczba neuronów,
  + ziarno generatora liczb losowych;
* Maszyna wektorów nośnych:
  + rodzaj maszyny wektorów nośnych,
  + funkcja jądra;
* K-najbliższych sąsiadów:
  + ilość sąsiadów,
  + metryka odległości,
  + algorytm szukania sąsiadów (wpływ na wydajność).

Parametry każdego z wymienionych klasyfikatorów tworzą przestrzeń w postaci wielowymiarowej hiperkostki, gdzie każdy parametr tworzy pojedynczy wymiar. W przestrzeni tej istnieje nieznana przez nas funkcja oceny przydatności klasyfikatora, którą chcemy maksymalizować. Rozmiar tej przestrzeni jest w większości przypadków zbyt duży, żeby problem ten rozwiązać metodą brutalnej siły, trenując klasyfikator w każdym punkcie – w szczególności, gdy mamy do czynienia z parametrami o ciągłych wartościach. Aby określić wartość optymalną, można skorzystać ze znanych metod numerycznych lub heurystyk. W szczególności bardzo interesujące jest zastosowanie algorytmów ewolucyjnych, czy optymalizacji rojem cząstek[[28]](#footnote-28).

W naszym przypadku postanowiliśmy zbadać przestrzeń parametrów za pomocą siatki punktów. Dla każdego parametru określamy kilka dozwolonych wartości, a następnie stosując iloczyn kartezjański generujemy wszystkie kombinacje, równo pokrywające badaną przestrzeń. Metoda ta nie znajduje optimum funkcji, jednak dane zebrane w ten sposób pozwalają na głęboką analizę zachowania klasyfikatorów. Ilość wygenerowanych w ten sposób konfiguracji jest bardzo duża. Dla sześciu parametrów mających po 5 wartości wynosi 15 625. Biorąc pod uwagę, że czas trenowania niektórych klasyfikatorów wynosił nawet 4 godziny, do przeszukania takiej przestrzeni potrzeba 62 500 godzin, czyli ponad 7 lat. Nasze testy zostały ograniczone do około 2500 eksperymentów, nadal jednak wymagałoby to ponad roku ciągłych obliczeń.

## System ewaluacji klasyfikatorów

Czas potrzebny na wytrenowanie wszystkich klasyfikatorów i porównanie ich charakterystyk jest stanowczo zbyt duży. Aby go zmniejszyć musimy zrównoleglić proces ich budowy. Mając do dyspozycji 100 rdzeni, moglibyśmy zmniejszyć potrzebny czas do niewiele ponad 4 dni. Postanowiliśmy stworzyć system rozproszony, który będzie koordynował trenowanie różnych klasyfikatorów i zapamiętywał wyniki ich testów. System ten działa na zasadzie podobnej do projektu Seti@Home[[29]](#footnote-29) – zawiera jeden węzeł serwera, który jest odpowiedzialny za przydzielanie zadań oraz wiele węzłów klienckich wykonujących obliczenia. Węzły klienckie komunikują się z serwerem centralnym poprzez sieć Ethernet, co umożliwia podłączenie do systemu dowolnej maszyny od chmur obliczeniowych przez wieloprocesorowe serwery po zwykłe komputery klasy PC, bez względu na rodzaj zainstalowanego systemu operacyjnego. System nie ma stałej konfiguracji – maszyny mogą podłączać i odłączać się w dowolnej chwili, co znacznie upraszcza zarządzanie i pozwala wykorzystać nieużywane zasoby obliczeniowe na przykład nocą, gdy większość komputerów jest zwykle wyłączona. Pojedyncza maszyna może zawierać wiele węzłów klienckich, najlepiej tyle ile rdzeni posiada.

Ethernet

Maszyna kliencka

Węzeł kliencki

Węzeł kliencki

Węzeł kliencki

Maszyna kliencka

Węzeł kliencki

Węzeł kliencki

Węzeł kliencki

Maszyna kliencka

Węzeł kliencki

Węzeł kliencki

Węzeł kliencki

Serwer centralny

Baza zadań

Generator zadań

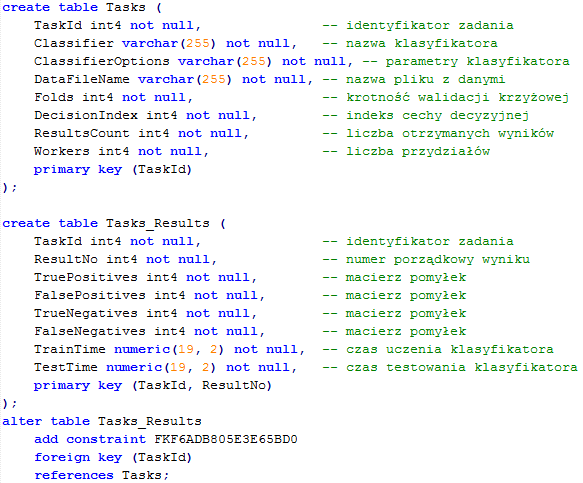
Analizator

wyników

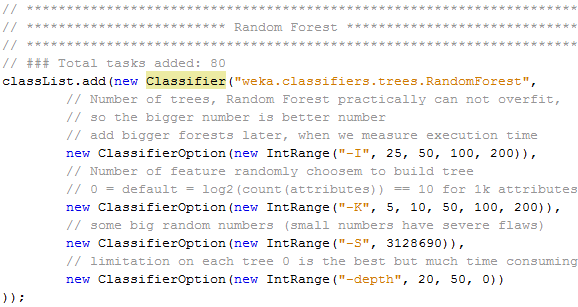
Ilustracja : Architektura systemu ewaluacji klasyfikatorów

## Generator zadań

Pierwszym krokiem wymaganym do uruchomienia systemu jest stworzenie listy zadań do wykonania. Pojedyncze zadanie składa się z nazwy klasyfikatora, zestawu parametrów i nazwy pliku z danymi. Trójki te są generowane na podstawie podanych przez użytkownika opisów badanych przestrzeni parametrów. Specyfikacje te są umieszczane bezpośrednio w kodzie niewielkiego programu pomocniczego, który następnie tworzy wszystkie kombinacje podanych opcji i zapisuje je w bazie danych.



Listing : Schemat bazy zadań



Listing : Przykładowa specyfikacja przestrzeni parametrów dla lasu losowego

## Serwer centralny

Następnym krokiem jest określenie protokołu komunikacji między serwerem a klientami oraz metody przydzielania zadań. Aby umożliwić dołączanie i odłączanie nowych węzłów w dowolnym momencie, serwer powinien być maszyną bezstanową, która pasywnie odpowiada na wysyłane przez klientów żądania przydziału zadań i zapisania wyników. W ten sposób nie musimy utrzymywać listy połączeń – po prostu nasłuchujemy na pojedynczym gnieździe na przychodzące komendy i je obsługujemy. Dzięki temu węzeł centralny jest bardzo prosty i przypomina serwer WWW obsługujący protokół HTTP. Natomiast najlepszą metodą przydziału zadań jest algorytm karuzelowy (ang. round-robin scheduling). Umieszcza on wszystkie zadania w kolejce i każdemu kolejnemu klientowi przydziela zadanie z czoła kolejki, usuwając je. Sytuacją wymagającą szczególnej uwagi jest awaria węzła podczas wykonywania zadania. Jej obsługa polega na tym, że pamiętamy ile razy każde zadanie było przydzielone. Gdy klient zgłasza gotowość, serwer szuka takiego zadania, które nie posiada jeszcze wyników, a ponadto ma najmniejszą liczbę przydziałów. Nawet jeżeli część węzłów ulegnie uszkodzeniu, zadania przez nie otrzymane zostaną w końcu wykonane przez inne węzły. Rozwiązanie to ma zaletę polegającą na tym, że jeżeli węzeł z jakiegoś powodu będzie przetwarzał zadanie bardzo wolno, ale nadal będzie działać, szybszy węzeł może także dostać to zadanie do wykonania i skończyć wcześniej.

## Węzły klienckie

Na maszynach klienckich uruchamiany jest program, który tworzy podaną ilość wątków stanowiących węzły klienckie. Węzły te posiadają pętlę, która pobiera zadanie z serwera, trenuje klasyfikator zgodny z opisem zadania, a następnie metodą walidacji krzyżowej (ang. cross-validation) testuje go. Wyniki testów w postaci macierzy pomyłek (ang. confusion matrix) oraz czasy trenowania i testowania są następnie odsyłane z powrotem do serwera. Węzeł kończy wykonanie w momencie, gdy serwer nie ma żadnych zadań do przydzielenia. Priorytet wątków można ustawić poniżej domyślnego dla danego systemu operacyjnego. Pozwoli to wykonywać obliczenia w tle, wykorzystując nieużywane cykle procesora, bez obciążania maszyny podczas pracy użytkownika. W ten sposób możemy do systemu podłączyć dowolną ilość komputerów osobistych, co znacząco zwiększa całkowitą moc obliczeniową.

## Analizator wyników

Gdy już wszystkie eksperymenty zostaną wykonane, możemy przystąpić do analizy zebranych wyników. Klasyfikator oceniając kolejne próbki ze zbioru testowego określa, czy według niego próbka jest reklamą, czy nie. Mając informację o prawdziwej klasie każdej próbki, odpowiedzi udzielone przez klasyfikator możemy podzielić na cztery kategorie:

Tabela : Macierz pomyłek

|  |  |  |
| --- | --- | --- |
| Ocena klasyfikatora \ Klasa | Reklama | Nie reklama |
| Reklama | Prawdziwy pozytyw (PP)[[30]](#footnote-30) | Fałszywy pozytyw (FP)[[31]](#footnote-31) |
| Nie reklama | Fałszywy negatyw (FN)[[32]](#footnote-32) | Prawdziwy negatyw (PN)[[33]](#footnote-33) |

Należy zwrócić uwagę na to, że klasyfikator może pomylić się na dwa sposoby. W przypadku fałszywego pozytywu oznacza treść jako reklamę, co w konsekwencji blokuje użytkownikowi dostęp do odwiedzanej przez niego strony internetowej lub jej fragmentu. W przypadku fałszywego negatywu reklama jest oznaczana jako treść i zostaje wyświetlona, co jest sprzeczne z celem działania systemu. Błędu pierwszego rodzaju są jednak znacznie bardziej dotkliwe, ponieważ powodują frustrację wśród użytkowników systemu. Jest to bardzo podobne do sytuacji, w której ważna wiadomość email zostaje oznaczona jako spam, nie jest wyświetlana w skrzynce odbiorczej i efektywnie nie zostaje odebrana. Użytkownicy reagują na taki błąd znacznie gorzej, niż na wyświetlenie kilku wiadomości ze spamem.

Na podstawie tych czterech wartości powstało wiele miar opisujących charakterystykę klasyfikatora. Moduł analizatora wyników odczytuje rezultaty wykonanych wcześniej zadań z bazy danych i oblicza różne miary. Ich wartości są używane do wytypowania optymalnego klasyfikatora. Ponadto sporządza wykresy ułatwiające analizę porównawczą różnych klasyfikatorów albo wpływu zmiany parametrów. Wśród badanych miar znajdują się:

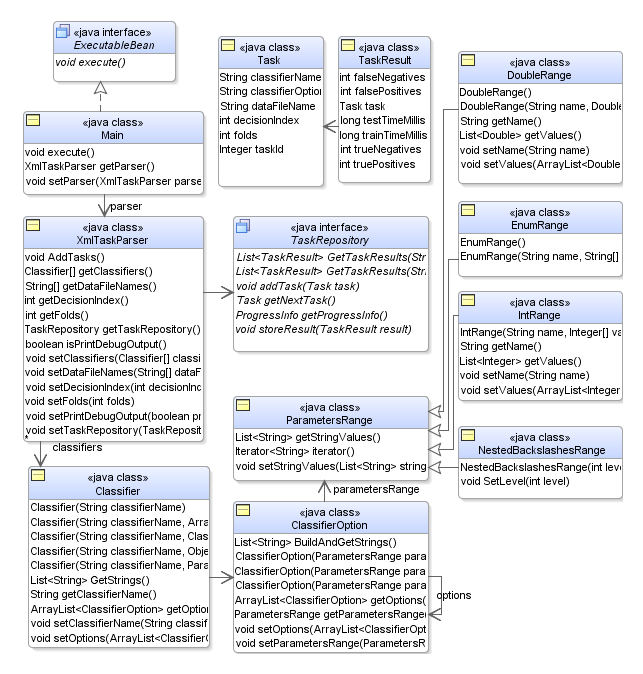
Tabela : Badane miary

|  |  |
| --- | --- |
| Nazwa miary | Wzór |
| Dokładność  (Accuracy) |  |
| Ważona dokładność  (Cost-sensitive accuracy) |  |
| Precyzja  (Precision) |  |
| Pełność  (Recall) |  |
| F-miara  (F-measure) |  |
| Ważona F-miara  (Cost-sensitive F-measure) |  |

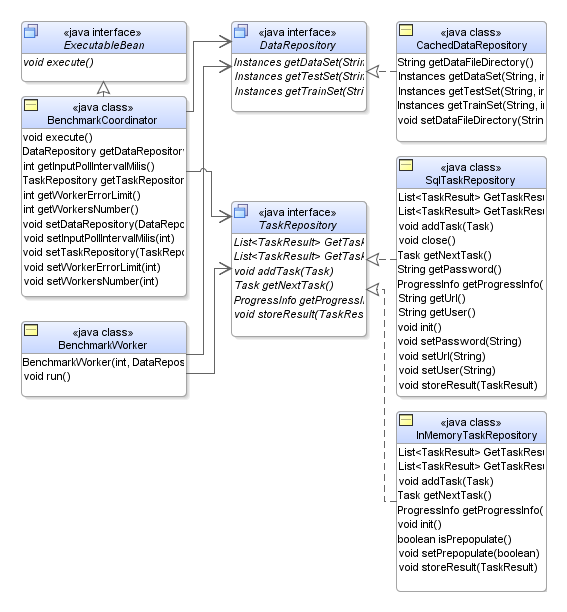
Każda z wymienionych miar opisuje inny fragment charakterystyki. Dokładność jest wartością najbliższą naszym oczekiwaniom, ponieważ bezpośrednio oddaje skuteczność klasyfikacji. Nie uwzględnia natomiast tego, że błędy FP są znacznie gorsze od FN. Właściwość tą można zamodelować poprzez dodanie wag, jednak dobór optymalnej wartości współczynnika alfa jest problematyczny. Na pewno chcemy, aby klasyfikator produkcyjny charakteryzował się precyzją zbliżoną do 100% (nie popełniał błędów typu FP), nawet kosztem mniejszej pełności. Celem stosowania F-miary jest złączenie obu tych miar, jednak aby odpowiednio oddać ważność precyzji musimy znów dobrać odpowiednią wagę. Ostateczna decyzja o wyborze najlepszej miary jest podejmowana ręcznie przez administratora systemu po przeanalizowaniu wszystkich zebranych danych. W naszym przypadku zdecydowaliśmy się na użycie ważonej F-miary ze współczynnikiem alfa równym 0,25. Dzięki temu faworyzujemy klasyfikatory osiągające dużą precyzję, przy zadowalającym poziomie pełności.

## Implementacja

Generator zadań operuje na opisach przestrzeni parametrów zawartych w klasie Classifier (z pakietu adblocker.taskparser). Obiekt tego typu opisuje całą przestrzeń parametrów dla pojedynczego klasyfikatora, której kształt składa się z hiperkostek określonych prze zbiór obiektów typu ClassifierOption. Przestrzeń parametrów często ma bardzo nieregularny kształt ze względu na parametry, które mogą być użyte jedynie wtedy, gdy inne parametry przyjmują określne wartości (np. różne funkcje przekształceń jądrowych używanych przez maszyny wektorów nośnych posiadają zupełnie inne zestawy parametrów). Hiperkostka ClassifierOption posiada zbiór obiektów ParametersRange, z których każdy opisuje przedział wartości jednego badanego parametru. Obiekty te odpowiadają kolejnym osiom przestrzeni. Ze względu na różne typy parametrów istnieją różne typy zasięgów – oparte o liczby całkowite, rzeczywiste i zbiory napisów. Na podstawie takiego opisu można wygenerować wszystkie interesujące punkty w przestrzeni, co robi klasa Classifier. Wszystkie wyniki są następnie umieszczane w repozytorium TaskRepository. Opisy badanych przestrzeni są zapisane bezpośrednio w kodzie klasy Main. Komponent XmlTaskParser umożliwia umieszczanie takich definicji w plikach XML, jednak nie obsługuje on skomplikowanych, wielokrotnych zagnieżdżeń występujących przy niektórych parametrach.



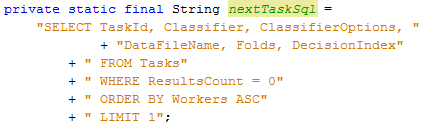
Ilustracja : Klasy generatora zadań



Ilustracja : Klasy węzła klienckiego

Wykonaniem zadań zajmuje się druga aplikacja wchodząca w skład modułu – węzeł kliencki. Klasa BenchmarkCoordinator, będąca punktem początkowym aplikacji, tworzy konfigurowalną liczbę wątków BenchmarkWorker. Ponadto synchronizuje dostęp do zasobów współdzielonych: repozytorium zadań oraz repozytorium danych. Każdy stworzony wątek pobiera zadanie z repozytorium, a następnie testuje z wykorzystaniem walidacji krzyżowej opisany w nim klasyfikator i zapisuje wyniki z powrotem w repozytorium. Jeżeli wątek podczas wykonania zadania zgłosi wyjątek, jest on wypisywany w oknie konsoli, a następnie wątek pobiera kolejne zadanie. W momencie, gdy liczba zgłoszonych wyjątków przekroczy wartość progową, wątek kończy wykonanie. Aplikację można uruchomić na dowolnej maszynie w dowolnym momencie. Żeby wykorzystać całą dostępną moc ilość wątków powinna odpowiadać fizycznej ilości rdzeni procesowa. Można ją także zatrzymać w dowolnym momencie bez żadnej szkody dla systemu, nawet po przydzieleniu zadania i w trakcie zapisywania wyników.

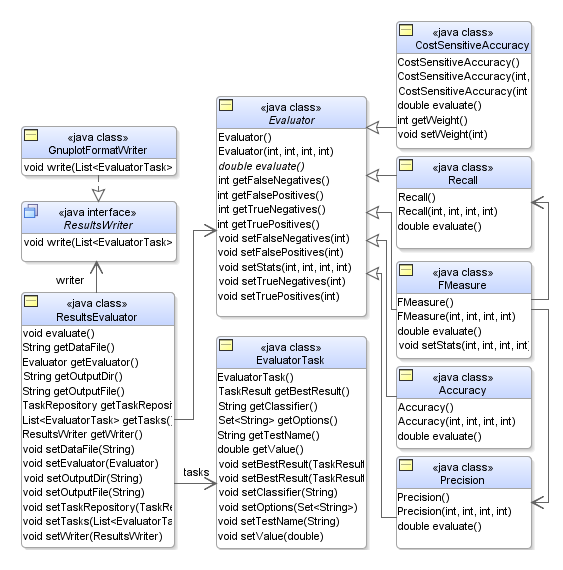
Istnieją dwie implementacje interfejsu TaskRepository: InMemoryTaskRepository trzymające dane w pamięci ulotnej i wykorzystywane podczas testów oraz SqlTaskRepository zapisujące wszystko trwale w bazie danych SQL. Repozytorium zawiera w sobie logikę odpowiedzialną za przydział następnego zadania. Dzięki temu węzeł centralny składa się jedynie z bazy danych i nie wymaga dodatkowej aplikacji koordynującej rozdział zadań. W przypadku SqlTaskRepository jest to osiągnięte poprzez użycie mechanizmu transakcji. Podczas pobierania zadania wybieramy takie, które nie zostało już wykonane oraz posiada najmniejszą liczbę przydziałów, a następnie od razu inkrementujemy ich liczbę. Daje nam to gwarancję, że zadanie to nie zostanie przydzielone ponownie przed przydzieleniem wszystkich pozostałych. Zadanie może być przydzielane wielokrotnie ze względu na awarię lub długi czas wykonania węzła, który je otrzymał.



Ilustracja : Zapytanie SQL zwracające następne zadanie

Badane klasyfikatory bardzo często działają na tym samym pliku z danymi. Aby wątki nie wczytywały danych wielokrotnie, opracowano interfejs DataRepository odpowiedzialny za dostarczenie odpowiednich zbiorów danych. Implementacja CachedDataRepository wczytuje zbiory z pliku podczas pierwszego żądania i zapamiętuje je w pamięci, dzięki czemu przy następnych żądaniach zwraca od razu referencję na gotowy zbiór, co znacznie przyspiesza wykonanie.

Ostatnim narzędziem wchodzącym w skład tego modułu jest analizator wyników ResultsEvaluator. Jego zadaniem jest znalezienie najlepszych klasyfikatorów według zadanej miary oceny Evaluator. Aplikacja wyszukuje po jednym najlepszym klasyfikatorze dla każdego zadania ewaluacji EvaluatorTask. Zadanie ewaluacji określa zbiór przeszukiwanych klasyfikatorów. Zwykle tworzonych jest kilka zadań odpowiadających różnym rodzajom klasyfikatorów lub temu samemu klasyfikatorowi, ale z dodatkowymi ograniczeniami na różne wartości parametrów. Najlepsze klasyfikatory są prezentowane przez interfejs ResultsWriter. Jego jedyna obecnie implementacja GnuplotFormatWriter rysuje wykresy w formacie gnuplot[[34]](#footnote-34).



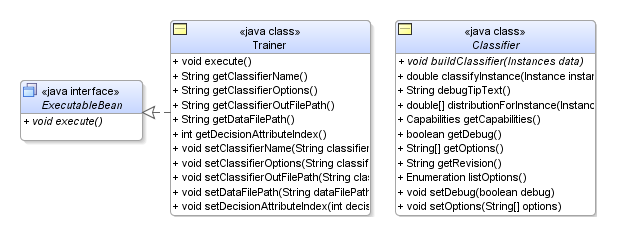
Ilustracja : Klasy analizatora wyników

# Moduł trenujący

Gdy mamy już przygotowane dane i wybrany klasyfikator wraz z parametrami możemy wytrenować ostateczny klasyfikator, który będzie działał w środowisku produkcyjnym. Zajmuje się tym moduł trenujący. Jest on najprostszym i najmniejszym modułem w całym systemie. Charakterystyka wytrenowanego klasyfikatora może różnić się od zmierzonej przez moduł ewaluacji. Dzieje się tak ponieważ podczas oceny klasyfikatorów używana jest walidacja krzyżowa, która dzieli zbiór danych na kilka części i dokładnie tyle samo razy buduje klasyfikator zostawiając jedną z nich jako zbiór testowy (w naszym przypadku jest to 1/5 zbioru). Ponadto końcowe wyniki są uśredniane. Trenowanie ostatecznego klasyfikatora wykorzystuje cały zbiór (25% więcej danych), co powinno pozytywnie wpłynąć na skuteczność klasyfikacji. Po stworzeniu klasyfikatora jest on zapisywany w postaci pliku binarnego.

## Implementacja

Od strony kodu moduł także nie jest skomplikowany. Metoda execute klasy Trainer stanowiąca punkt startowy aplikacji tworzy klasyfikator na podstawie podanych w pliku konfiguracyjnym parametrów i zapisuje go do pliku. Jej implementacja zajmuje dosłownie 4 linie.



Ilustracja : Kluczowe klasy modułu trenującego

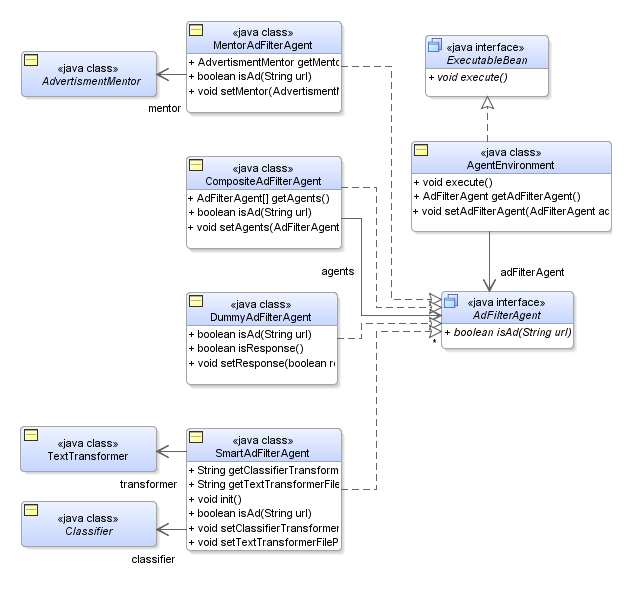
# Moduł agent produkcyjnego

Ostatnim etapem jest stworzenia agenta, który będzie odpowiadał na pytanie, czy przedstawiony adres URL jest reklamą czy nie. Podstawowa wersja agenta wykorzystuje metody sztucznej inteligencji w postaci klasyfikatora zbudowanego przez moduł trenujący. Używa także transformatora przygotowanego przez moduł przetwarzania wstępnego, ponieważ musi napływające adresy URL przetłumaczyć na wektory cech zrozumiałe dla klasyfikatora. Istnieją także inne rodzaje agentów. Do dyspozycji mamy również wersję, która odpowiada zgodnie z listą wyrażeń regularnych EasyList, co pozwala naszemu systemowi emulować znane i sprawdzone rozwiązanie. Trzecim i ostatnim typem agenta jest agent kompozytowy pozwalający połączyć ze sobą oba. Uznaje on odnośnik za reklamę, jeżeli dowolny z agentów podrzędnych tak uzna. W ten sposób jesteśmy w stanie dodatkowo zwiększyć skuteczność systemu. Patrząc z perspektywy agenta inteligentnego, dostajemy gwarancję, że nie pomyli on się przy adresach pokrytych przez listę. Natomiast patrząc z perspektywy agenta opartego o listę ulepszamy go o możliwość automatycznego wyłapania adresów podobnych do opisanych za pomocą wyrażeń regularnych.

Agent jest osadzany w odpowiednim środowisku. W przypadku systemu produkcyjnego jest to serwer pośredniczący monitorujący ruch przeglądarki internetowej użytkownika końcowego. Jeżeli agent uzna, że adres URL żądania HTTP jest reklamą, serwer nie pobiera zasobu z sieci i odsyła pustą odpowiedź. Istnieje także wersja środowiska, umożliwiająca uruchomienie agenta z linii poleceń. Pozwala ona sprawdzić jego zachowanie w odniesieniu do różnych adresów i jest używana w celach testowych.

## Implementacja

Klasa AgentEnvironment jest punktem startowym aplikacji. W pętli czeka na wpisanie przez użytkownika adresu, a następnie odpytuje komponent agenta implementujący interfejs AdFilterAgent, czy jest on reklamą i wyświetla wynik. Rodzaj i parametry agenta można określić w pliku konfiguracyjnym. Implementacją docelową jest SmartAdFilterAgent. Napływające do niego adresy tłumaczy na instancję klasy SparseInstance za pomocą obiektu TextTransformer stworzonego przez moduł przetwarzania wstępnego. Próbkę w tej postaci podaje na wejście klasyfikatora zapisanego przez moduł trenowania i zwraca wynik.



Ilustracja : Kluczowe klasy modułu agenta

# Wyniki

## Zebrane dane

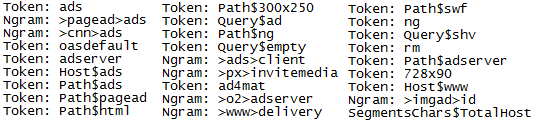
Dane na podstawie których przeprowadzono badania zebrano poprzez logowanie ruchu przeglądarek internetowych członków projektu. Odwiedzanymi stronami były popularne polskie portale. W ciągu kilku sesji uzbierano 55 202 adresy URL – 52 914 strony z treścią i 2288 reklam. Następnie dokonano ekstrakcji cech, w tym (po wstępnej filtracji):

* 7695 symboli
* 11 500 symboli z informacją o komponencie URL
* 21 735 bigramów
* 24 196 trigramów

Stosując heurystykę przyrostu informacji i algorytm CFS do selekcji cech, a następnie algorytm grupujący do redukcji ilości przykładów ograniczono rozmiar zbioru. Stworzono w ten sposób kilka różnych zestawów:

* 100 cech (CFS 1000, Przyrost informacji 100), 7293 próbek
* 200 cech (CFS 1000, Przyrost informacji 200), 10065 próbek
* 500 cech (CFS 1000, Przyrost informacji 500), 11597 próbek
* 1000 cech (CFS 1000), 12099 próbek
* 2000 cech (CFS 2000), 12411 próbek
* 5000 cech (CFS 5000), 13576 próbek

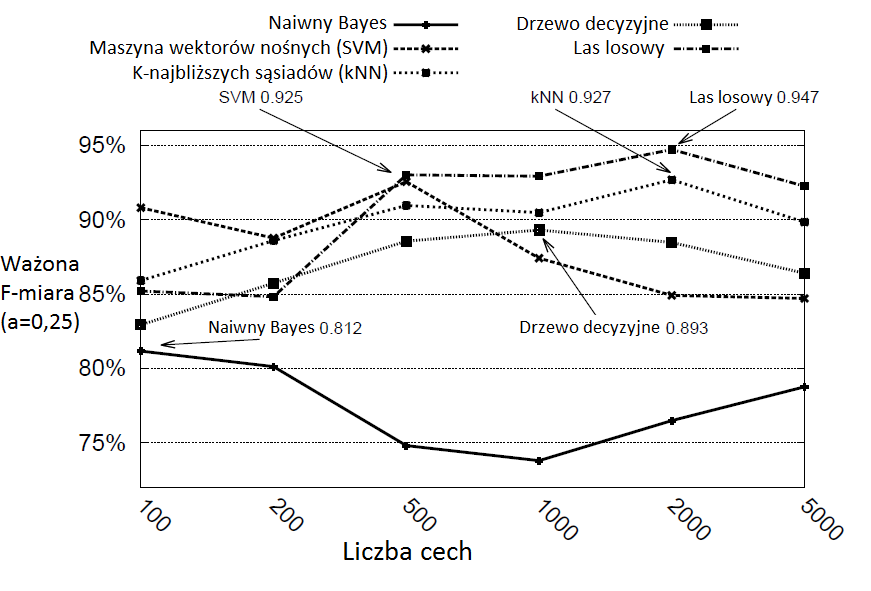
Wśród pozostawionych cech większość stanowią symbole powstałe ze słowa *ads*. Występują także charakterystyczne dla reklam rozmiary. Symbole najczęściej zawierają informacje o komponencie URL z którego pochodzą. Długość nazwy hosta jest ważna informacją.



Ilustracja : Najlepsze wg. algorytmu przyrostu informacji cechy

## Skuteczność systemu

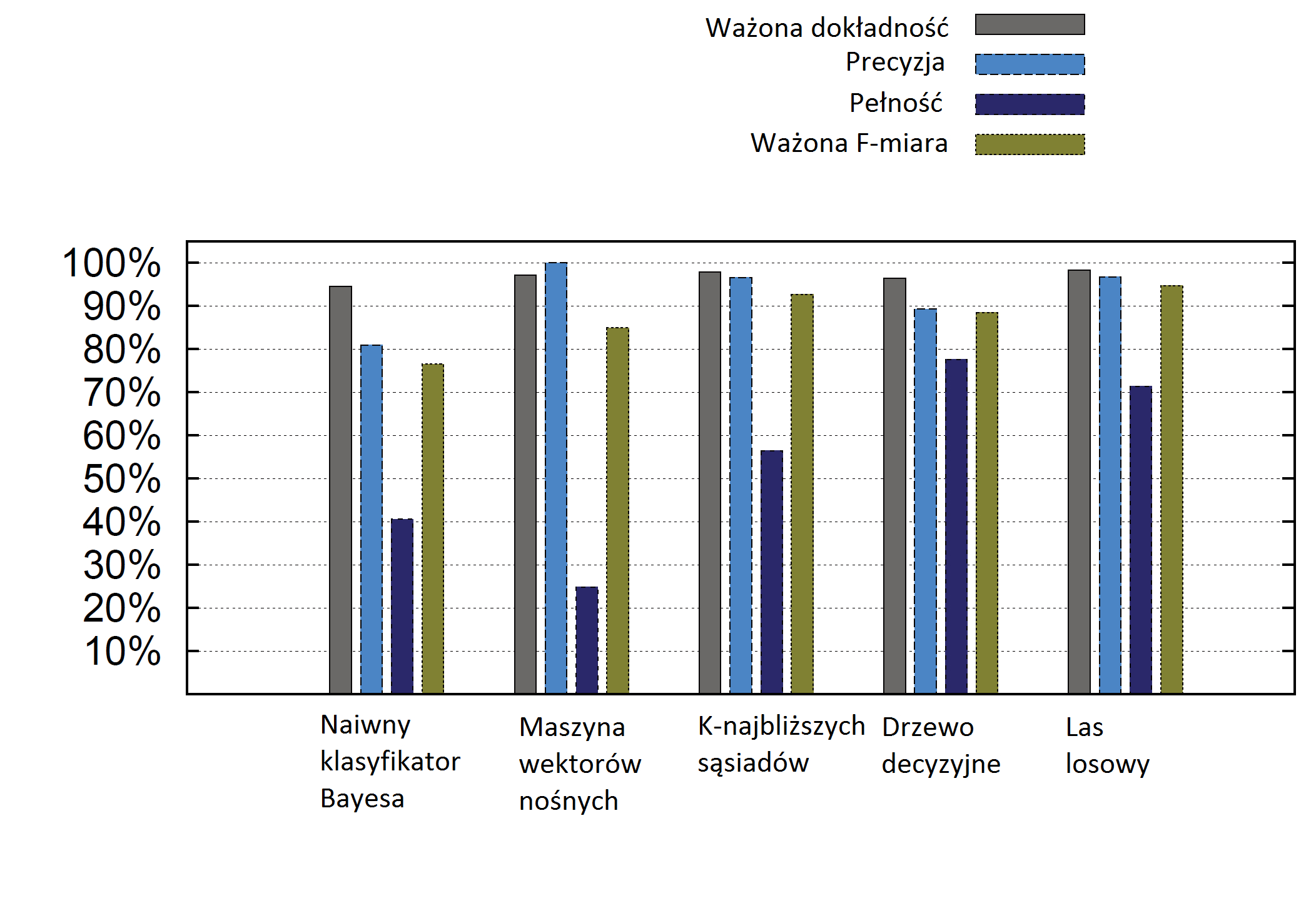
Skuteczność systemu została zbadana poprzez wytrenowanie ponad 2500 klasyfikatorów różnych typów, z różnymi konfiguracjami parametrów i na różnych zbiorach danych. Pierwszym etapem badań było określenie optymalnej ilości cech w zbiorze danych. Eksperymenty wykazały, że najlepsze rezultaty są osiągane przy 2000 cech. Zwiększanie ich ilości powoduje spadek skuteczności prawie wszystkich klasyfikatorów.



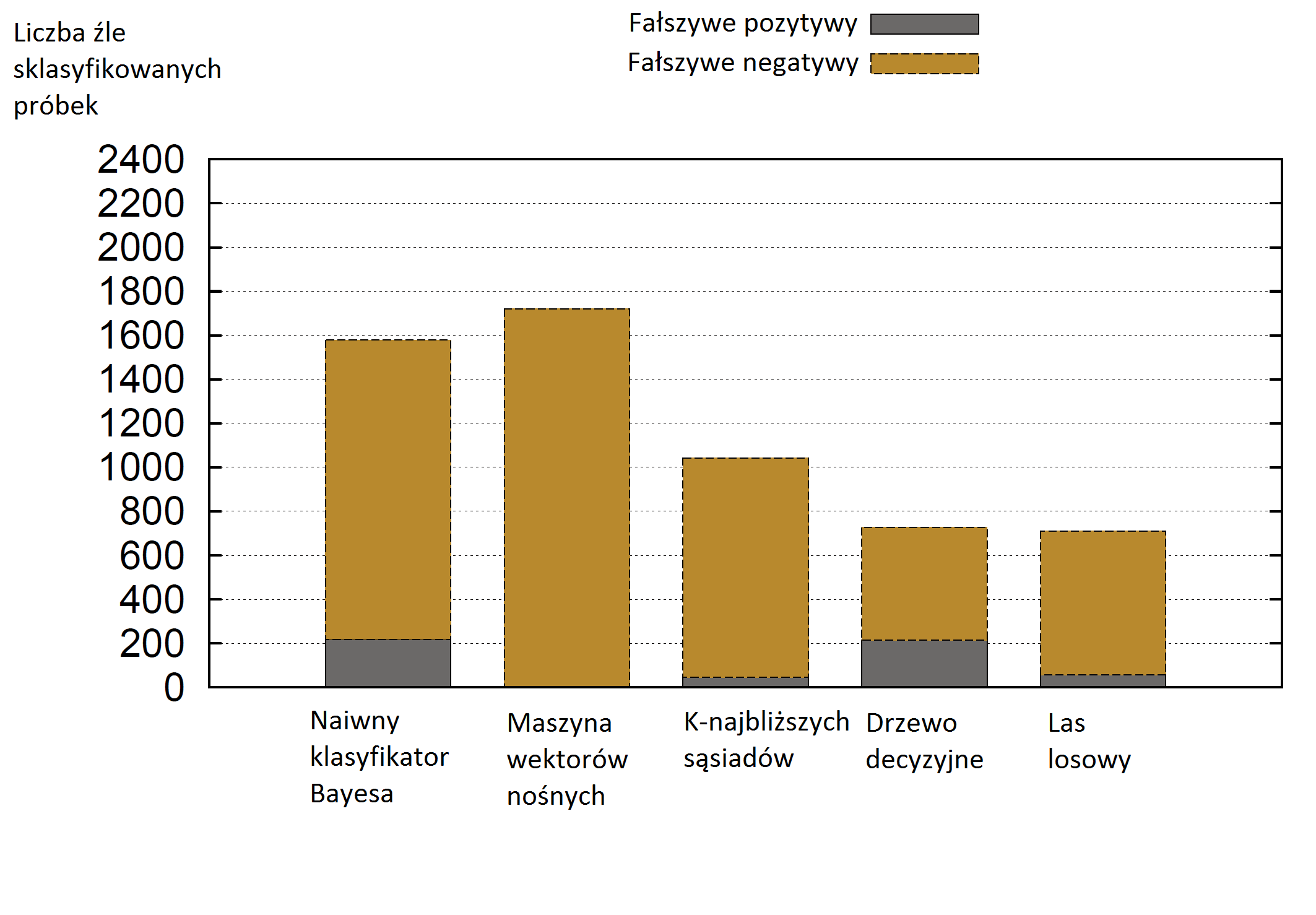
Ilustracja : Ocena klasyfikatorów względem liczby cech

Drugim etapem było wyłonienie w zbiorze klasyfikatorów zbudowanych na 2000 cechach najlepszych klasyfikatorów danego typu. Użyto do tego ważonej F-miary ze współczynnikiem alfa wynoszącym 0,25. Wartości różnych miar tych klasyfikatorów są te są widoczne na ilustracji poniżej. Ostatni etap polegał na wybraniu najlepszego klasyfikatora. Okazał się nim las losowy. Jego ustawienia to:

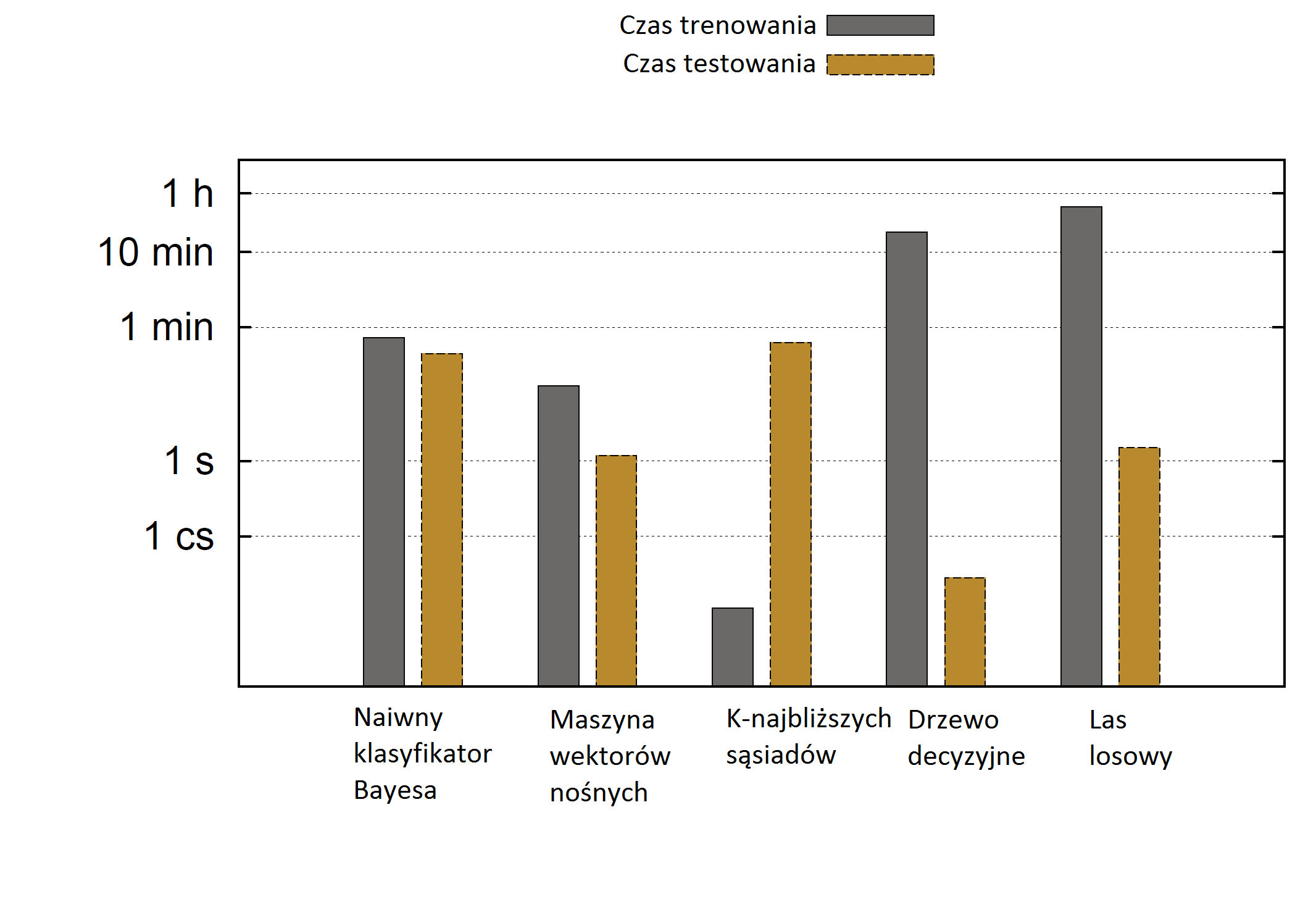
* liczba drzew: 200,
* liczba losowych cech użytych do budowy drzewa: 50,
* maksymalna wysokość drzewa: 50.



Ilustracja : Ocena badanych klasyfikatorów – 2000 cech



Ilustracja : Liczba błędów badanych klasyfikatorów – 2000 cech



Ilustracja : Czas trenowania i testowania badanych klasyfikatorów – 2000 cech

Klasyfikator ten ma 98,71% dokładności (710 błędów na zbiorze 55 202 próbek), znajduje 71,42% reklam (654 z 2288 pominął) oraz ma precyzję rzędu 96,69% (56 z 1690 niepoprawnych decyzji typu reklama). Ostatecznie osiągnął 94,71% ważonej F-miary. Jego dodatkową zaletą jest krótki czas odpowiedzi – dla całego zbioru wyniósł niecałe półtora sekundy (1,492s). Duża część błędów powstaje poprzez podział zbioru danych podczas walidacji krzyżowej. Jeżeli mała grupa rzadko występujących adresów reklam przejdzie w całości ze zbioru trenującego do testowego klasyfikator nic o nich nie wie i nie ma szans ocenić ich poprawnie. W każdym przebiegu walidacji aż 11 040 próbek z 55 202 jest przeznaczana do testowania. Szansa na wystąpienie takiego zjawiska jest dosyć duża. Z tego względu klasyfikator wytrenowany na całym zbiorze może osiągać lepsze wyniki. Dokładne omówienie wyników oraz ich analizę można znaleźć w pracy Krzysztofa Kamińskiego [[2](#Krz13)].

# Podsumowanie

Stworzony system poprawianie rozpoznaje prawie 99% adresów URL, przy czym wychwytuje ponad 71% reklam i popełnia zaledwie 0,1% błędów typu fałszywy pozytyw. Są to bardzo obiecujące wyniki, które – przy dalszym rozwoju systemu – umożliwiają jego komercyjne zastosowanie. Prezentowane rozwiązanie dodatkowo umożliwia głęboką analizę zachowania klasyfikatorów wszystkich typów względem różnych parametrów, różnych algorytmów selekcji i ekstrakcji cech oraz wielu innych ustawień całego toru eksploracji danych. Jest to ogromne źródło wiedzy, które po dogłębnym zbadaniu pozwoli osiągnąć jeszcze lepszą skuteczność. Rozproszona architektura systemu pozwala na jego skalowanie i przeprowadzanie eksperymentów, które na jednej maszynie zajęłyby wiele lat. Dzięki temu możemy przetestować tysiące konfiguracji i wyłonić najlepszą z nich. Otwarta, komponentowa struktura całego rozwiązania pozwala w każdej chwili dodawać nowe klasyfikatory, nowe ekstraktory cech, algorytmy selekcji cech oraz rozszerzyć lub wymienić dowolny istniejący komponent.

## Kierunki rozwoju

### Duże zbiory danych

Badany zbiór danych zawiera ponad 55 tysięcy próbek. Mimo, że jest to duża liczba, to jednak daleko jej do wielkości całej sieci WWW. Aby system działał na wszystkich stronach internetowych – nie tylko popularnych polskich portalach – należy opracować metodę pozyskiwania i przygotowania danych, która tak olbrzymie ilości będzie w stanie przetworzyć w rozsądnym czasie. Można tutaj zastosować algorytmy grupowania, aby zbiór wszystkich adresów URL wstępnie podzielić na klika rodzajów adresów i każdą z tych grup przetwarzać oddzielnie, co zmniejszy objętość danych przetwarzanych przez każdy klasyfikator. Może się okazać, że niektóre grupy stworzone w ten sposób w ogóle nie zawierają reklam i mogą być od razu klasyfikowane jako treść. Badanie charakterystyk stworzonych w ten sposób grup może dostarczyć wielu cennych informacji. Możliwe, że w zależności od grupy, inne cechy będą miały znacznie i dodatkowo pozwoli to ograniczyć ilość cech.

### Dokładniejsza selekcja cech

Kolejnym problemem jest ulepszenie algorytmów selekcji cech. Stosowana przez nas heurystyka przyrostu informacji może odrzucać wiele cech, których wartości wydają się być losowe, jednak po złożeniu ich z innymi cechami niosą kluczową dla klasyfikacji informację. Z drugiej strony algorytmy oceniające grupy cech są zbyt złożone, aby je wykorzystać dla bardzo dużych rozmiarów danych. Rozproszenie ich wykonania na odpowiednio wiele maszyn mogłoby ten problem rozwiązać. Innym podejściem jest tworzenie losowych podzbiorów cech i uruchamianie algorytmów tego typu na odpowiednio małych zbiorach, a następnie konsolidacja otrzymanych wyników.

### Wstępna klasyfikacja oparta o zachowanie użytkowników

Do wstępnej klasyfikacji używamy istniejących rozwiązań co obarcza system ich błędami. W przypadku komercyjnego zastosowania można wykorzystać do tego opisaną wcześniej heurystykę polegającą na ocenie zasobu poprzez to w jaki sposób użytkownicy z nim postępują. Zasoby często blokowane powinny być oznaczane jako reklamy, co z kolei powinno skutkować przebudowaniem klasyfikatora. Proces może być w całości automatyczny, dzięki czemu system będzie uczył się w czasie rzeczywistym na podstawie ruchu tysięcy, a może nawet milionów użytkowników. Przy blokowaniu zasoby użytkownik może dodatkowo podać przyczynę blokowania: reklama, treść irytująca, treść nieodpowiednia. Pozwoli to dodatkowo blokować strony, które nie są reklamami, ale podobnie jak one utrudniają poruszanie się po sieci. Idąc dalej tym tropem, system może zostać użyty do rozpoznawania nie tylko reklam, ale dowolnie określonej treści.

### Klasyfikacja na podstawie kontekstu

Kolejnym dużym zagadnieniem jest analiza treści strony internetowej w celu określenia kontekstu występowania zasobu. System przy niewielkich modyfikacjach jest w stanie sprostać temu zadaniu. Umożliwi to wprowadzenie wielu cech, które mogą okazać się bardzo pomocne, a nawet kluczowe podczas klasyfikacji. Ilość informacji zawarta w strukturze strony internetowej jest ogromna. Reklamy zdradzają się bardzo często już samą nazwą identyfikatora lub klasy elementu, w którym są osadzone. Miejsce występowania elementu też jest bardzo charakterystyczne – podłużny obrazek u góry strony prawdopodobnie jest banerem reklamowym. Mając na uwadze wyścig zbrojny ze strony agencji reklamowych, cechy te bardzo trudno ukryć.

### Klasyfikacja na podstawie treści

Oprócz cech opartych o kontekst można także próbować zastosować techniki rozpoznawania treści: tekstu, obrazu czy nawet mowy w odniesieniu do filmów ze ścieżką dźwiękową. Metody te są kopalnią nowych cech, jednak reklamy są na tyle zróżnicowane, że ich zastosowanie będzie możliwe dopiero w dalszym horyzoncie czasu, gdy będziemy dysponować odpowiednio dużą wiedzą i mocą obliczeniową.

### Heurystyka wyboru najlepszego klasyfikatora

Podczas wybory optymalnego klasyfikatora, wykorzystaliśmy metodę przeszukiwania przestrzeni parametrów opartą o siatkę punktów. W tym miejscu można zastosować znane algorytmy optymalizacji, w tym algorytmy ewolucyjne, symulowane wyżarzanie, rój cząstek itd. Aby osiągnąć lepszy wynik od prezentowanego przez nas wystarczy zastosować dowolną metodę optymalizacji lokalnej, która z sztywnego punktu na siatce poprowadzi do maksimum lokalnego. Przy stosowaniu tych metod trzeba jednak mieć na uwadze duży czas potrzebny na obliczanie wartości pojedynczego punktu. Może to być nawet kilka godzin, dlatego wybrana metoda powinna umożliwiać rozproszenie wykonania, tak aby na raz obliczać wartości wielu punktów.

Bibliografia

x

|  |  |
| --- | --- |
| 1 | Januszewski M. System filtracji reklam internetowych. 2012. |
| 2 | Kamiński K. Comparative study of machine-learning algorithms for the detection of ads on web pages. 2013. |
| 3 | Szczepański P, Wiśniewski A. An Automated Framework with Application to Study URL Based Online Advertisements Detection. Journal of the Applied Mathematics, Statistics and Informatics. 2013. |
| 4 | Line B. SITEIMPULSE. [Internet]. [dostęp 4 czerwca 2013]. <http://www.siteimpulse.com/blog/internetowe-formy-reklamowe-rodzaje-formaty-i-rozmiary-nie-tylko-banerow/>. |
| 5 | Szyperski C. Component Software: Beyond Object-Oriented Programming. Pearson Education; 2002. |
| 6 | Kan MY, Thi HON. Fast webpage classification using URL features. Proceedings of the 14th ACM international conference on Information and knowledge management. 2005 325-326. |
| 7 | Fayyad U, Irani K. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. Proceedings of the International Joint Conference on Uncertainty in AI. 1993 1022-1027. |
| 8 | Shih LK, Karger DR. Using URLs and Table Layout for Web Classiﬁcation Tasks. Proceedings of the 13th international conference on World Wide Web. 2004 193 - 202. |
| 9 | He Y, Liu K, Moran S. Choosing the Best Bayesian Classifier: An Empirical Study. IAENG International Journal of Computer Science, 36. 2009 322–331. |

x

Spis ilustracji i tabel

[Ilustracja 1: Schemat inteligentnego systemu rozpoznawania reklam 14](#_Toc359193863)

[Ilustracja 2:Przykład konfiguracji komponentu agenta w Spring IOC 15](#_Toc359193864)

[Ilustracja 3: Kluczowe klasy modułu pozyskiwania danych 18](#_Toc359193865)

[Ilustracja 4: Kluczowe klasy modułu przetwarzania wstępnego 28](#_Toc359193866)

[Ilustracja 5: Klasy ekstraktorów 29](#_Toc359193867)

[Ilustracja 6: Klasy filtrów (transformacji) danych 30](#_Toc359193868)

[Ilustracja 7: Architektura systemu ewaluacji klasyfikatorów 34](#_Toc359193869)

[Ilustracja 8: Klasy generatora zadań 39](#_Toc359193870)

[Ilustracja 9: Klasy węzła klienckiego 40](#_Toc359193871)

[Ilustracja 10: Zapytanie SQL zwracające następne zadanie 41](#_Toc359193872)

[Ilustracja 11: Klasy analizatora wyników 42](#_Toc359193873)

[Ilustracja 12: Kluczowe klasy modułu trenującego 43](#_Toc359193874)

[Ilustracja 13: Kluczowe klasy modułu agenta 45](#_Toc359193875)

[Ilustracja 14: Najlepsze wg. algorytmu przyrostu informacji cechy 46](#_Toc359193876)

[Ilustracja 15: Ocena klasyfikatorów względem liczby cech 47](#_Toc359193877)

[Ilustracja 16: Ocena badanych klasyfikatorów – 2000 cech 48](#_Toc359193878)

[Ilustracja 17: Liczba błędów badanych klasyfikatorów – 2000 cech 48](#_Toc359193879)

[Ilustracja 18: Czas trenowania i testowania badanych klasyfikatorów – 2000 cech 49](#_Toc359193880)

[Listing 1: Fragment strony http://www.wp.pl – reklamy znajdują się w blokach div oznaczonych jako adv 9](#_Toc359193881)

[Listing 2: Adresy URL odwiedzone kolejno podczas jednej sesji 10](#_Toc359193882)

[Listing 3: Schemat bazy zadań 35](#_Toc359193883)

[Listing 4: Przykładowa specyfikacja przestrzeni parametrów dla lasu losowego 35](#_Toc359193884)

[Równanie 1: Przyrost informacji cechy f na zbiorze danych X 24](#_Toc359193885)

[Równanie 2: Funkcja oceny przydatności algorytmu CFS 24](#_Toc359193886)

[Równanie 3: Ważona częstotliwość występowania symbolu t w dokumencie d ze zbioru dokumentów D 25](#_Toc359193887)

[Tabela 1: Cechy na podstawie przykładowych adresów 21](#_Toc359193888)

[Tabela 2: Macierz pomyłek 37](#_Toc359193889)

[Tabela 3: Badane miary 37](#_Toc359193890)

Zawartość płyty DVD

Płyta zawiera skompilowane pliki wykonywalne wraz z kodem źródłowym całego systemu, a także wykorzystane dane. System składa się z wielu projektów, z których każdy posiada własny katalog o strukturze zgodnej z projektem typu Maven Java Application.

Katalog główny:

* /.setting – katalog z ustawieniami projektu Netbeans całego systemu,
* /agents – projekt modułu agentów,
* /benchmark – projekt modułu ewaluacji klasyfikatorów,
* /common – projekt zasobów wspólnych,
* /data – zestawy danych,
* /database – schemat bazy danych modułu ewaluacji klasyfikatorów,
* /lists – listy wyrażeń regularnych,
* /preprocess – projekt modułu przetwarzania wstępnego,
* /results – projekt analizatora wyników,
* /target – katalog z plikami wykonywalnymi (wersja ostateczna),
* /taskparser – projekt generatora zadań,
* /text – listy adresów URL,
* /train – projekt modułu trenowania,
* /urlproxylogger – projekt serwera pośredniczącego,
* .project – plik projektu Netbeans całego systemu,
* assembly.xml – plik definicji budowy ostatecznych plików wykonywalnych,
* pom.xml – plik projektu Maven całego systemu.

Katalog projektu:

* /agents/src – kod źródłowy projektu,
* /agents/target – pliki wykonywalne projektu,
* /agents/.classpath – plik projektu Netbeans,
* /agents/.project – plik projektu Netbeans,
* /agents/nbactions.xml – plik projektu Netbeans,
* /agents/nb-configuration.xml – plik projektu Netbeans,
* /agents/pom.xml – plik projektu Maven.

Wszystkie pliki wykonywalne znajdują się w katalogu /target/adblocker-1.0-assembly:

* /data – zestawy danych,
* /lists – listy wyrażeń regularnych,
* /text – listy adresów URL,
* agents.config.xml – plik konfiguracji modułu agenta produkcyjnego,
* benchmark.config.xml – plik konfiguracji modułu ewaluacji klasyfikatorów,
* preprocess.config.xml – plik konfiguracji modułu przetwarzania wstępnego,
* results.config.xml – plik konfiguracji analizatora wyników,
* taskparser.config.xml – plik konfiguracji generatora zadań,
* taskreposittory.config.xml – plik konfiguracji bazy danych,
* tasks.config.xml – plik konfiguracji generowanych zadań,
* train.config.xml – plik konfiguracji modułu trenowania,
* urlproxylogger.config.xml – plik konfiguracji serwera pośredniczącego,
* Agents-1.0.jar – plik wykonywalny modułu agenta produkcyjnego,
* Benchmark-1.0.jar – plik wykonywalny modułu ewaluacji klasyfikatorów,
* Common-1.0.jar – biblioteka wspólna,
* Preprocess-1.0.jar – plik wykonywalny modułu przetwarzania wstępnego,
* Results-1.0.jar – plik wykonywalny analizatora wyników,
* taskparser-1.0.jar – plik wykonywalny generatora zadań,
* Train-1.0.jar – plik wykonywalny modułu trenowania,
* Urlproxylogger-1.0.jar – plik wykonywalny serwera pośredniczącego.

Płyta nie zawiera surowych plików z adresami URL, które zostały zebrane poprzez urządzenia mobilne, ze względu na poufność formatu tych plików. Usunięto także komponent odpowiedzialny za wczytywanie ich wczytywanie. Jednakże zbudowane na ich podstawie zbiory danych znajdujące się w katalogu data zawierają te adresy, co umożliwia powtórzenie wszystkich eksperymentów.

Konfiguracja i uruchamianie systemu

**Wymagane aplikacje:**

* Uruchomienie systemu
  + Środowisko wykonania Javy (JRE) <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
  + Dowolna baza danych SQL  
    np. <http://www.postgresql.org/>
* Kompilacja
  + Zestaw narzędzi Java (JDK)  
    <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
  + Środowisko programisty (IDE) obsługujące Maven  
    np. <https://netbeans.org/> (zawiera Maven)
  + Maven  
    <http://maven.apache.org/>

Przed uruchomieniem systemu należy stworzyć bazę danych za pomocą skryptu /database/schema.sql oraz utworzyć użytkownika, który będzie posiadał prawa odczytu i zapisu do stworzonych tabel. Następnie z katalogu /target można uruchamiać kolejne moduły systemu.

Aby skompilować system należy przy pomocy narzędzia Maven zbudować projekt /pom.xml. Można to zrobić poprzez IDE obsługujące to narzędzie, importując wskazany plik jako projekt. W przypadku środowiska Netbeans można skorzystać z gotowego pliku projektu /.project. Narzędzie Maven samodzielnie pobierze wszystkie wymagane biblioteki podczas próby kompilacji (wymagane połączenie z Internetem).

System wykorzystuje paradygmat programowania komponentowego oraz kontener komponentów Spring IOC[[35]](#footnote-35). Punktem startowym każdej stworzonej aplikacji jest stworzenie wszystkich obiektów zgodnie ze skazanym plikiem konfiguracyjnym w formacie XML, który można edytować za pomocą dowolnego edytora tekstu. Pozwala to wymienić każdy wykorzystywany przez aplikację obiekt na inny (oczywiście implementujący wymagany zbiór interfejsów). Można także zmieniać wszystkie wartości pól obiektów – w tym referencji, co pozwala dowolnie komponować tworzone obiekty. W efekcie można w ramach systemu wykorzystać dowolne klasyfikatory napisane w ramach pakietu eksploracji danych WEKA[[36]](#footnote-36), dowolne algorytmy selekcji cech dostępne w tym samym pakiecie oraz wiele innych komponentów, które nie zostały użyte podczas naszych badań.

**Serwer pośredniczący**

* Plik wykonywalny: urlproxylogger-1.0.jar
* Plik konfiguracyjny: urlproxylogger.config.xml

Serwer pośredniczący stworzony przez Michała Januszewskiego i opisany dokładnie w [[1](#Mic12)]. Umożliwia blokowanie reklam w dowolnej przeglądarce internetowej – wystarczy odpowiednio skonfigurować jego ustawienia. Właściwość port określa port na którym serwer nasłuchuje. Ustawienia agenta podajemy we właściwości adAgent. Serwer pozwala także śledzić ruch użytkownika i zapisywać wszystkie odwiedzone adresy URL do wskazanego w relativeFilePrefix katalogu. Ważne, aby przy takim użyciu właściwość adAgent ustawić na DummyAgent z polem response ustawionym na false. W ten sposób serwer nie będzie blokował reklam i umożliwi swobodne przeglądanie sieci WWW.

**Moduł przetwarzania wstępnego**

* Plik wykonywalny: Preprocess-1.0.jar
* Plik konfiguracyjny: preprocess.config.xml

Aplikacja tworzy zbiór danych (plik docelowy to data/temp/temp.arff). Wykorzystuje w tym celu źródło adresów URL CompositeTextDao złożone z FileDao, które z kolei wczytuje adresy z plików z katalogu /text. Metoda wstępnej klasyfikacji jest określona jako RegExpListMentor i korzysta z list wyrażeń znajdujących się w katalogu /lists. Właściwość extractor obiektu Preprocessor zawiera referencję na komponent odpowiedzialny za ekstrakcję cech. Jest to CompositeUrlExtractor złożony z obiektów typu: SegmentsMissingExtractor, TokenCountExtractor, SegmentLengthExtractor, UrlNumericCountExtractor, SimpleTokenExtractor, N\_gramExtractor. Natomiast pole filter opisuje filtr używany do transformacji danych MultiFilter będący kompozytem obiektów typu: FrequentAttribute, AttributeSelection, MergeSameInstancesFilter.

**Moduł doboru klasyfikatorów – generacja zadań**

* Plik wykonywalny: taskparser-1.0.jar
* Plik konfiguracyjny: taskparser.config.xml (importuje taskrepository.config.xml)

Narzędzie tworzy wpisy w bazie danych opisujące zadania do wykonania. Zadania są opisane w kodzie narzędzia. W pliku konfiguracyjnym możemy zmienić nazwy plików z danymi i krotność walidacji krzyżowej generowanych zadań. W pliku taskrepository.config.xml, który jest współdzielony przez kilka aplikacji, należy podać rodzaj oraz ustawienia bazy danych. Domyślnie repozytorium zadań korzysta z implementacji InMemoryTaskRepository, która jest ulotną bazą trzymaną w pamięci procesu używaną do testów. Można skorzystać z implementacji SqlTaskRepository, która łączy się z bazą danych SQL. Przykładowa konfiguracja znajduje się wewnątrz komentarza na początku pliku konfiguracyjnego.

**Moduł doboru klasyfikatorów – wykonanie**

* Plik wykonywalny: Benchmark-1.0.jar
* Plik konfiguracyjny: benchmark.config.xml (importuje taskrepository.config.xml)

Aplikacja ta umożliwia wykonanie zadań zapisanych w repozytorium zadań. Domyślnie jest to InMemoryTaskRepository zawierające 100 zadań dotyczących klasyfikatora k-najbliższych sąsiadów. Zalecane jest skonfigurowanie repozytorium tak, aby korzystało z bazy danych SQL oraz uruchomienie generatora zadań przed uruchomieniem tej aplikacji. Narzędzie tworzy obiekt BenchmarkCoordinator, który tworzy workersNumber wątków będących klientami rozproszonego systemu wykonania. Każdy wątek pobiera zadanie z repozytorium, tworzy i testuje klasyfikator, po czym zapisuje jego wyniki w repozytorium.

**Moduł doboru klasyfikatorów – analiza**

* Plik wykonywalny: Results-1.0.jar
* Plik konfiguracyjny: results.config.xml (importuje taskrepository.config.xml)

Aplikacja pozwala analizować wyniki wykonania. Przed jego użyciem należy uruchomić narzędzie Benchmark, tak aby w repozytorium znajdowały się wyniki. Poprzez właściwość evaluator można ustawić kryterium wyłaniania najlepszego klasyfikatora (Accuracy, CostSensitiveAccuracy, FMeasure, Precision lub Recall). Typy badanych klasyfikatorów określamy poprzez właściwość tasks – każdy wpis odpowiada jednemu wyszukiwaniu najlepszego klasyfikatora ograniczonym do typu i opcjonalnie podanych wartości parametrów. Właściwość writer pozwala wybrać rodzaj prezentacji wyników (obecnie tylko GnuplotFormatWriter).

**Moduł trenowania**

* Plik wykonywalny: Train-1.0.jar
* Plik konfiguracyjny: train.config.xml

Narzędzie tworzy klasyfikator produkcyjny. Właściwość classifierOutFilePath pozwala określić ścieżkę zapisu. Typ klasyfikatora i jego parametry określamy odpowiednio poprzez classifierName i classifierOptions. Trzeba także podać ścieżkę do pliku z danymi – dataFilePath. Właściwość decisionAttributeIndex określa indeks cechy będącej klasą próbki.

**Moduł agenta**

* Plik wykonywalny: Agents-1.0.jar
* Plik konfiguracyjny: agents.config.xml

Pozwala na przetestowanie stworzonego agenta za pomocą adresów URL wpisywanych z linii komend. Właściwość adFilterAgent pozwala zdefiniować rodzaj i ustawienia agenta. Istnieją cztery implementacje:

* CompositeAdFilterAgent – agent kompozytowy,
* DummyAdFilterAgent – agent testowy odpowiadający zawsze tak samo,
* MentorAdFilterAgent – agent wykorzystujący listę wyrażeń regularnych,
* SmartAdFilterAgent – agent wykorzystujący klasyfikator.

1. ang. Hypertext Transfer Protocol – protokół komunikacji między przeglądarką, a serwerem WWW [↑](#footnote-ref-1)
2. ang. Uniform Resource Locator – ujednolicony adresu zasobu [↑](#footnote-ref-2)
3. ang. HyperText Markup Language – język zapisu dokumentów w sieci WWW [↑](#footnote-ref-3)
4. Plik z animacją odtwarzaną przez Adobe Flash Player [↑](#footnote-ref-4)
5. ang. Cascading Style Sheets – arkusze styli z opisem wyglądu elementów (rozmiar, kolor itp.) [↑](#footnote-ref-5)
6. Pobrano dnia 21.05.2013 o godzinie 15:34 przy użyciu przeglądarki Firefox [↑](#footnote-ref-6)
7. Problem ten został opisany w [[8](#Law)] [↑](#footnote-ref-7)
8. ang. Multipurpose Internet Mail Extensions – identyfikator formatu przesyłanego pliku [↑](#footnote-ref-8)
9. http://adblockplus.org/en/firefox [↑](#footnote-ref-9)
10. http://code.google.com/p/adblockforchrome/ [↑](#footnote-ref-10)
11. http://easylist.adblockplus.org/en/ [↑](#footnote-ref-11)
12. https://easylist.adblockplus.org/blog/2011/09/01/easylist-statistics:-august-2011 [↑](#footnote-ref-12)
13. http://adblockplus.org/en/subscriptions [↑](#footnote-ref-13)
14. http://www.cs.waikato.ac.nz/ml/weka/ [↑](#footnote-ref-14)
15. http://www.r-project.org/ [↑](#footnote-ref-15)
16. http://www.mathworks.com/products/matlab/ [↑](#footnote-ref-16)
17. http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html [↑](#footnote-ref-17)
18. http://www.worldwidewebsize.com/ – dane z 06.05.13 [↑](#footnote-ref-18)
19. ang. Text Mining [↑](#footnote-ref-19)
20. Nie bezpośrednio. Mogą natomiast zawierać zakodowaną reprezentację: %20, jednak kombinacje tego typu nie są używane w nazwie hosta. Jeżeli już występują to zwykle jako wartość parametrów zapytania HTTP. [↑](#footnote-ref-20)
21. [[6](#Min)] [↑](#footnote-ref-21)
22. Dokładny opis znajduje się w RFC 3986 – http://tools.ietf.org/html/rfc3986#section-3 [↑](#footnote-ref-22)
23. Kodowanie pozwalające zapisać dane binarne za pomocą znaków ASCII. [↑](#footnote-ref-23)
24. Należy zauważyć, że kolejność tokenów występujących w nazwie hosta jest odwrócona: mamy com>domaintools, a nie domaintools>com [↑](#footnote-ref-24)
25. Adres reklamy pochodzącej z http://atemda.com/JSAdservingSP.ashx?[...] [↑](#footnote-ref-25)
26. Ang. Term Frequency – Inverse Document Frequency [↑](#footnote-ref-26)
27. http://www.cs.waikato.ac.nz/ml/weka/arff.html [↑](#footnote-ref-27)
28. Więcej informacji o podejściu tego typu można znaleźć w [[9](#Stu)] [↑](#footnote-ref-28)
29. http://setiathome.berkeley.edu/ [↑](#footnote-ref-29)
30. ang. True positive (TP) [↑](#footnote-ref-30)
31. ang. False positive (FP) [↑](#footnote-ref-31)
32. ang. False negative (FN) [↑](#footnote-ref-32)
33. ang True negative (TN) [↑](#footnote-ref-33)
34. http://www.gnuplot.info/ [↑](#footnote-ref-34)
35. <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html> [↑](#footnote-ref-35)
36. <http://www.cs.waikato.ac.nz/ml/weka/> [↑](#footnote-ref-36)