|  |  |
| --- | --- |
| 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 marek takich jak McDonald’s, Allegro, czy gazeta.pl. Mimo wielu sukcesów, postanowiłem – zgodnie ze swoimi marzeniami z dzieciństwa – tworzyć gry komputerowe, więc 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 i podjąć pracę w firmie Google, gdzie będę mógł dalej 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 nieraz przytłaczające, szczególnie dla posiadaczy urządzeń mobilnych, którzy płacą za pobierane dane. Praca 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 w tym celu 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. This 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](#_Toc357696637)

[1.1. Cel pracy 6](#_Toc357696638)

[1.2. Zakres 7](#_Toc357696639)

[1.3. Powiązane prace naukowe i publikacje 7](#_Toc357696640)

[Rozdział 2. Filtrowanie reklam w sieci WWW 8](#_Toc357696641)

[2.1. Po czym rozpoznać reklamę? 9](#_Toc357696642)

[2.2. Obecne rozwiązania 11](#_Toc357696643)

[2.3. Metody inteligentne 12](#_Toc357696644)

[Rozdział 3. Inteligentny system wykrywania 13](#_Toc357696645)

[3.1. Pozyskiwanie danych 15](#_Toc357696646)

[3.2. Przetwarzanie wstępne 17](#_Toc357696647)

[3.2.1. Ekstrakcja cech 17](#_Toc357696648)

[3.2.2. Selekcja cech 21](#_Toc357696649)

[3.2.3. Dodatkowa obróbka 23](#_Toc357696650)

[3.3. Dobór klasyfikatorów 24](#_Toc357696651)

[3.3.1. Zakres badań 24](#_Toc357696652)

[3.3.2. System ewaluacji klasyfikatorów 24](#_Toc357696653)

[3.3.3. Generacja zadań 25](#_Toc357696654)

[3.3.4. Koordynacja wykonania 25](#_Toc357696655)

[3.3.5. Wykonanie 25](#_Toc357696656)

[3.3.6. Analiza wyników 25](#_Toc357696657)

[3.4. Trenowanie 27](#_Toc357696658)

[3.5. Agent produkcyjny 28](#_Toc357696659)

[Rozdział 4. Implementacja 29](#_Toc357696660)

[Rozdział 5. Wyniki 34](#_Toc357696661)

[5.1. Zebrane dane 34](#_Toc357696662)

[5.2. Skuteczność systemu 34](#_Toc357696663)

[Rozdział 6. Podsumowanie 39](#_Toc357696664)

[Bibliografia 41](#_Toc357696665)

[Spis ilustracji 44](#_Toc357696666)

[Dodatek A. Zawartość płyty CD 46](#_Toc357696667)

[Dodatek B. Konfiguracja i uruchamianie systemu 47](#_Toc357696668)

# 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. Dzieje się tak ponieważ – 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 pracy

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.

## Zakres

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

Niniejsza praca 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 i implementacja architektury systemu, co zostało opisane w tej pracy naukowej. Powstały jeszcze dwie prace naukowe:

* O proxy – Januszewski
* O wynikach – Kamiński

Ponadto powstała publikacja naukowa:

* An Automated Framework with Application to Study URL Based Online Advertisements Detection

# Filtrowanie reklam w sieci WWW

Poruszając się po sieci WWW wykonujemy szereg zapytań protokołu HTTP[[1]](#footnote-1). Protokół ten pozwala zażądać od wskazanego serwera przesłania zasobu znajdującego 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 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, co spowodowało znaczny spadek ilości reklam dystrybuowanych tą metodą. Istnieje także odmiana wyświetlająca okienko pod oknem z treścią (ang. pop-under).

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” - 120x60 px, „box” - 125x125 px, „baner” – 400x50 px) do reklam zakrywających prezentowaną treść („toplayer”, „interstitial”). Często zawierają animacje, których ruch dodatkowo odwraca uwagę użytkownika od czytanej treści. Ponadto niektóre starają się upodobnić do okienek systemu operacyjnego, co dodatkowo dezorientuje i wprowadza w błąd.

## 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 poziomym obrazkiem u góry strony, istnieje spora szansa, 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

Mimo tego, analiza kontekstu elementu na stronie internetowej jest zadaniem dosyć ciężkim i czasochłonnym, w przeciwieństwie do analizy adresów URL, które są w znacznie krótsze i prostsze. Adresy te 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, dlatego 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 : Przykładowe adresy URL zasobów pobranych przez przeglądarkę podczas sesji

Kolejne cechy możemy uzyskać dopiero po wysłaniu żądania do serwera i odebraniu zasobu. Pozwoli nam to uzyskać informacje o rozmiarze i typie MIME[[7]](#footnote-7) 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 odpowiedzialny za tłumaczenie nazw hostów 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 istniejące agencje jest bardzo proste i wymaga jedynie stworzenia nowej subdomeny, z której będą wysyłane reklamy, co nie jest trudne i można robić to dowolnie często.

Bardziej złożone rozwiązania, w tym najpopularniejsze takie jak Adblock Plus[[8]](#footnote-8) czy AdBlock[[9]](#footnote-9), opierają się o listy specjalnych wyrażeń regularnych. Poszczególne wyrażenia – zwane filtrami – są porównywane z adresami URL żądań i w przypadku dopasowania żądanie jest odrzucane. Istnieje także zbiór filtrów działających jako biała lista, pozwalających na bezpieczne pobranie zasobu. Listy wyrażeń są tworzone i utrzymywane najczęściej niezależnie od programów. Najpopularniejszą jest EasyList[[10]](#footnote-10), która jest używana przez ponad 12 milionów użytkowników[[11]](#footnote-11). Zawiera ona reguły blokujące większość dużych i znanych serwerów reklamowych, a także wyłącza te mniejsze działające w anglojęzycznej części sieci WWW. Ponadto lista ta posiada różne rozszerzenia[[12]](#footnote-12). 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 lub 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 reguły i zależności między cechami pozwalające rozróżnić reklamy od treści. W przypadku przekierowań hostów używamy tylko nazwy domeny, używając list wyrażeń regularnych dysponujemy jedynie adresem URL. Natomiast wykorzystując algorytmy uczenia maszynowego, możemy definiować dowolne atrybuty, także te związane z 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 zbiór trenując składający się z setek tysięcy, a nawet milionów deskryptorów zasobów[[13]](#footnote-13), jest w stanie odkryć znacznie więcej subtelnych i nawet 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. Wymagają za to odpowiedniego zbioru uczącego, który jest listą zasobów wraz z etykietą, czy zasób jest reklamą, czy też nie. Można go uzyskać na kilka sposobów:

* Automatycznie za pomocą istniejących programów do blokowania reklam
* Przy pomocy użytkowników programu, dając im możliwość blokowania i odblokowywania poszczególnych zasobów
* Zatrudniając ludzi do kategoryzowania zawartości zasobów sieci WWW (co nadal jest prostsze niż pisanie reguł)

Każdy z nich ma swoje wady i zalety, które są omówione w następnym rozdziale.

# Inteligentny system wykrywania

Naszym celem jest stworzenie systemu do filtrowania reklam, który będzie wykorzystywał metody uczenia maszynowego, przy czym ograniczamy się do rozpoznawania reklam na podstawie adresów URL. Produktem końcowym jest agent, który po podaniu adresu URL udzieli binarnej odpowiedzi, czy adres jest reklamą. Agenta można następnie umieścić na serwerze pośredniczącym, który monitoruje ruch użytkownika lub bezpośrednio w przeglądarce internetowej w postaci wtyczki i odrzucać odpowiednie żądania protokołu HTTP. W tym rozdziale zostanie szczegółowa architektura oraz proces jego tworzenia.

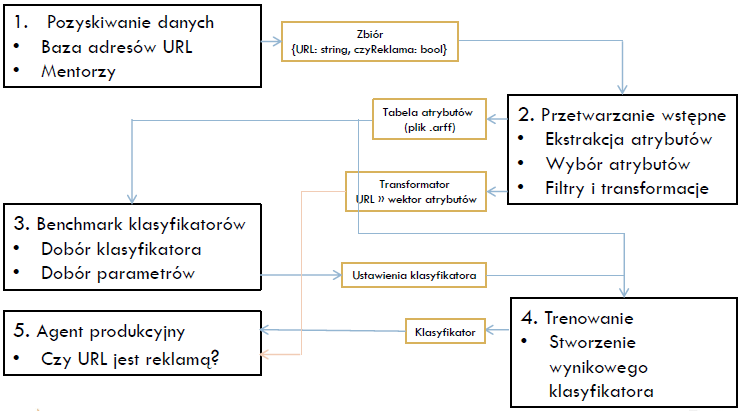
Cały system składa się z pięciu modułów. Pierwszy zajmuje się pozyskiwaniem danych. Większość algorytmów uczenia maszynowego wymaga istnienia zbioru trenującego, na którym algorytm się uczy. Produktem tego modułu jest lista adresów URL wraz z informacją, czy jest to reklama. Głównym problemem jest dokonanie tej wstępnej klasyfikacji na ogromnych 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 algorytmów. 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 metrykę, na podstawie której zostanie wyłoniony najbardziej odpowiedni 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.



Ilustracja : Schemat inteligentnego systemu rozpoznawania reklam

## Pozyskiwanie danych

Sieć WWW jest olbrzymim źródłem danych. Na chwilę obecną szacuje się, że indeks wyszukiwarki internetowej Google obejmuje ponad czterdzieści pięć miliardów różnych stron internetowych[[14]](#footnote-14). 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 loguje 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 xxx adresów URL. Ponadto system udostępnia interfejs RawTextDao, który pozwala na wykorzystanie dowolnych innych źródeł danych.

Zadaniem znacznie trudniejszym od zebrania listy adresów URL jest ich wstępna klasyfikacja na reklamy lub treść. Stajemy przed swojego rodzaju paradoksem jajka i kury, ponieważ tworzymy system do rozpoznawania reklam, który już na wstępnie wymaga od nas umiejętności dokonania takiej klasyfikacji. Z sytuacji tej można wyjść na trzy sposoby. Najbardziej oczywistym jest ręczna klasyfikacja zebranych adresów. Przy zbiorach danych wielkości kilkuset tysięcy adresów jest to jednak 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 w 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 daniu użytkownikowi możliwości blokowania i odblokowywania poszczególnych zasobów. Mechanizm ten i tak musi być wbudowany w system, ponieważ nawet najdokładniejszy klasyfikator czasami może się pomylić. Dlatego ważne jest, aby użytkownik mógł odblokować treść, która została odrzucona jako reklama. Mając informację o żądaniach blokowanych i odblokowywanych przez użytkowników możemy bez problemu określić ich klasę. Rozwiązanie to ma także szereg innych zalet. Uwidacznia i pozwala szybko reagować na błędy systemu. Jeżeli użytkownicy często ręcznie kontrolują jakiś zasób, można go dodać do zbioru trenującego i zaktualizować klasyfikator. Może się to dziać nawet automatycznie. Metoda ta jest bardzo obiecująca, gdy system już działa, jednak nie jest odpowiednia przy rozruchu systemu, ponieważ jakość klasyfikacji byłaby zadowalająca dopiero po pewnym czasie od wdrożenia. Użytkownicy mogą nie mieć tyle cierpliwości.

Do wstępnej klasyfikacji postanowiliśmy wykorzystać listę filtrów EasyList wraz z polskim suplementem. W ten sposób natychmiast uzyskaliśmy klasyfikację dla całego zebranego uprzednio zbioru adresów. System poprzez interfejs Mentor pozwala na podłączenie dowolnej innej metody wstępnej klasyfikacji. Fizycznym rezultatem tego etapu przetwarzania jest plik CSV[[15]](#footnote-15), który posiada dwie kolumny: adres URL oraz wartość logiczną 1 dla reklam i 0 dla treści.

## Przetwarzanie wstępne

### Ekstrakcja cech

Adresy URL w postaci ciągów znaków nie są zrozumiałe dla algorytmów klasyfikacji. Oczekują one opisania wszystkich próbek za pomocą wektorów wartości numerycznych, tak aby długość wektora odpowiadała ilości badanych cech, a poszczególne wartości były pewnym odwzorowaniem tych cech w przestrzeń liczb rzeczywistych.

Zagadnieniem ekstrakcji cech z ciągów znaków zajmuje się dziedzina eksploracji tekstu[[16]](#footnote-16). 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 pokrywają się z nazwa 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 typowego tekstu. Z jednej strony są znacznie prostsze, ponieważ zwykle nie zawierają znaków interpunkcyjnych (średnik zdarza się rzadko), ale z drugiej nie mogą zawierać spacji[[17]](#footnote-17), przez co bardzo często mamy do czynienia ze zbitkami słownymi typu: googleads, pagead itp. Problem ten można próbować rozwiązać wykorzystując słownik i teorię entropii próbując łamać napotkane symbole na mniejsze, jednak uzyskany wzrost skuteczności klasyfikacji nie jest duży[[18]](#footnote-18). Wiąże się to z tym, że dużo takich związków jest nazwami własnymi, charakterystycznymi dla serwerów reklamowych. Ponadto słowa mogą występować 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. Dopuścić można także luźne n-gramy, które oprócz sąsiadów, dopuszczają wszystkie kombinacje, w których zachowana jest kolejność występowania symboli zgodna z oryginalną. Ze względu na 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[[19]](#footnote-19) 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 skrótowce i niezrozumiałe ciągi znaków) oraz wartości tych parametrów (nierzadko będące liczbami lub nawet zakodowanymi ciągami BASE64[[20]](#footnote-20)). Na stworzonym przez nas zbiorze trenującym ich liczba wyniosła xxx, co bezpośrednio przekłada się na liczbę cech. Ponadto z symboli tych są tworzone n-gramy, co dodatkowo powoduje eksplozję kombinatoryczną. Adresy URL składają się jednak z co najwyżej kilku-kilkunastu słów, dlatego większość cech dla danego adresu ma wartość równą zero. Ważne jest, aby macierz cech zapisywać w postaci rzadkiej, ponieważ umożliwia to pomieszczenie 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 xxx[[21]](#footnote-21) – 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.

System pozwala także na dodawanie dowolnych innych cech, wystarczy zaimplementować interfejs TextExtractor. Istnieje możliwość wprowadzenia cech opartych nie o adres URL, ale o nagłówki HTTP i kontekst elementu wyświetlanego na stronie. Wymaga to niewielkiej modyfikacji fazy zbierania danych, tak aby zapamiętywać także odpowiedź serwera wraz z pobieranym dokumentem html. Prześledźmy jeszcze proces ekstrakcji cech na przykładzie dwóch podobnych 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[[22]](#footnote-22)** | 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[[23]](#footnote-23)), 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 bardzo dużą liczbę. 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[[24]](#footnote-24). 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, co zwiększy skuteczność klasyfikatora na zbiorze testowym. 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[[25]](#footnote-25), które próbują estymować ich przydatność. Pozwala to wyeliminować niepotrzebne i zachować jedynie użyteczne cechy. 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 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 grupy 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 to najczęściej ma wartość równą zero, co w konsekwencji uniemożliwia 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 (Fayyad i Irani, 1993), 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 cech 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.

## Dobór 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 nowy 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.

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[[27]](#footnote-27) – 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

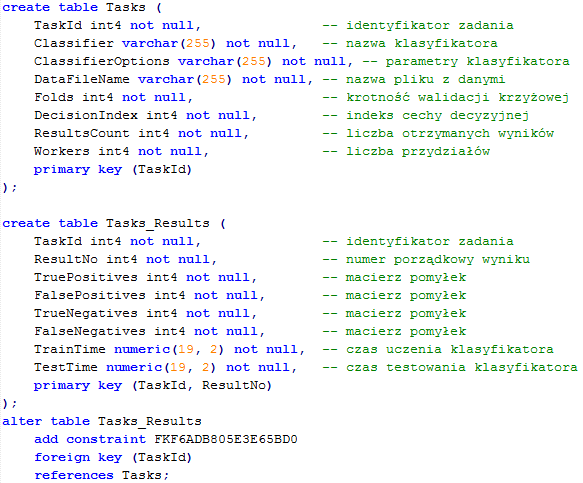
Analizator

wyników

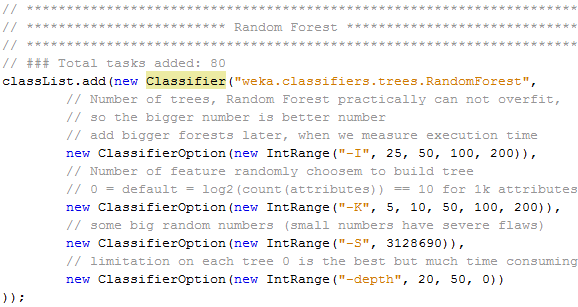
Ilustracja : Architektura systemu ewaluacji klasyfikatorów

### 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[[28]](#footnote-28). Umieszcza on wszystkie zadania w kolejce i każdemu kolejnemu klientowi przydziela zadanie z czoła kolejki, usuwając je. Kolejność w jakiej zadania są umieszczane w kolejce w naszym przypadku nie ma znaczenia. 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[[29]](#footnote-29) testuje go. Wyniki testów w postaci macierzy pomyłek[[30]](#footnote-30) 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)[[31]](#footnote-31) | Fałszywy pozytyw (FP)[[32]](#footnote-32) |
| Nie reklama | Fałszywy negatyw (FN)[[33]](#footnote-33) | Prawdziwy negatyw (PN)[[34]](#footnote-34) |

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 metryk opisujących charakterystykę klasyfikatora. Moduł analizatora wyników odczytuje rezultaty wykonanych wcześniej zadań z bazy danych i oblicza stosowne miary, które są używane do wytypowania optymalnego klasyfikatora. Ponadto sporządza wykresy ułatwiające analizę porównawczą różnych klasyfikatorów albo wpływu zmiany parametrów. Wśród badanych metryk znajdują się:

Tabela : Badane metryki

|  |  |
| --- | --- |
| Nazwa metryki | 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 trochę inny fragment charakterystyki, przez co nie można jednoznacznie określić, który klasyfikator jest najlepszy. Dokładność jest metryką 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 metryk, jednak aby odpowiednio oddać ważność precyzji musimy znów dobrać odpowiednią wagę. Ostateczna decyzja o wyborze najlepszego klasyfikatora jest podejmowana ręcznie przez administratora systemu po przeanalizowaniu wszystkich zebranych danych, uwzględniając osobiste doświadczenie.

## Trenowanie

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 próbek), co powinno pozytywnie wpłynąć na skuteczność klasyfikacji. Po stworzeniu klasyfikatora jest on zapisywany w postaci pliku binarnego.

## Agent produkcyjny

Ostatnim etapem jest stworzenia agenta, który będzie odpowiadał na pytanie, czy przedstawiony adres URL jest reklamą czy nie.

SmartAdAgent – wykorzystuje zbudowany klasyfikator

AdblockerAgent – wykorzystuje filtry Adblockera

CompositeAdAgent – rozwiązanie hybrydowe

* Środowkisko
  + Proxy
  + Linia poleceń

# Implementacja

5 stron

# Wyniki

5 stron

## Zebrane dane

## Skuteczność systemu

# Podsumowanie

2 strony

O wyścigu zbrojnym – dodanie nowych cech – polozenia nie oszukasz

Bibliografia

Fayyad, U. i Irani, K. (1993). Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. *Proceedings of the International Joint Conference on Uncertainty in AI*, 1022-1027.

Spis ilustracji

Ilustracja 1: Schemat inteligentnego systemu rozpoznawania reklam 14

Zawartość płyty CD

Konfiguracja i uruchamianie systemu

1. ang. Hypertext Transfer Protocol – protokół komunikacji między przeglądarką, a serwerem WWW [↑](#footnote-ref-1)
2. ang. Uniform Resource Locator – ujednolicony zapis 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. ang. Multipurpose Internet Mail Extensions – identyfikator formatu przesyłanego pliku [↑](#footnote-ref-7)
8. http://adblockplus.org/en/firefox [↑](#footnote-ref-8)
9. http://code.google.com/p/adblockforchrome/ [↑](#footnote-ref-9)
10. http://easylist.adblockplus.org/en/ [↑](#footnote-ref-10)
11. https://easylist.adblockplus.org/blog/2011/09/01/easylist-statistics:-august-2011 [↑](#footnote-ref-11)
12. http://adblockplus.org/en/subscriptions [↑](#footnote-ref-12)
13. Każdy zasób jest opisany przez wektor wartości, z których każda mierzy kolejną cechę zasobu. [↑](#footnote-ref-13)
14. http://www.worldwidewebsize.com/ – dane z 06.05.13 [↑](#footnote-ref-14)
15. Comma Separated Values – plik tekstowy reprezentujący macierz, której pola są zwykle oddzielone przecinkami [↑](#footnote-ref-15)
16. ang. Text Mining [↑](#footnote-ref-16)
17. 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-17)
18. Dodać referencję do artykułu! [↑](#footnote-ref-18)
19. Dokładny opis znajduje się w RFC 3986 – http://tools.ietf.org/html/rfc3986#section-3 [↑](#footnote-ref-19)
20. Kodowanie pozwalające zapisać dane binarne za pomocą znaków ASCII. [↑](#footnote-ref-20)
21. [↑](#footnote-ref-21)
22. Należy zauważyć, że kolejność tokenów występujących w nazwie hosta jest odwrócona: mamy com>domaintools, a nie domaintools>com [↑](#footnote-ref-22)
23. Adres reklamy pochodzącej z http://atemda.com/JSAdservingSP.ashx?[...] [↑](#footnote-ref-23)
24. ang. overfitting [↑](#footnote-ref-24)
25. ang. Feature Selection [↑](#footnote-ref-25)
26. Ang. Term Frequency – Inverse Document Frequency [↑](#footnote-ref-26)
27. http://setiathome.berkeley.edu/ [↑](#footnote-ref-27)
28. ang. round-robin scheduling [↑](#footnote-ref-28)
29. ang. cross-validation [↑](#footnote-ref-29)
30. ang. confusion matrix [↑](#footnote-ref-30)
31. ang. True positive (TP) [↑](#footnote-ref-31)
32. ang. False positive (FP) [↑](#footnote-ref-32)
33. ang. False negative (FN) [↑](#footnote-ref-33)
34. ang True negative (TN) [↑](#footnote-ref-34)