Rok akademicki 2014/2015

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Marcin M

Klasyfikacja list na stronach internetowych z uwzględnieniem atrybutów wizualnych

Opiekun pracy:

dr inż. Piotr A

Ocena . . . . . .

. . . . . .

Podpis Przewodniczącego

Komisji Egzaminu Dyplomowego

Kierunek: Informatyka

Specjalność: Inżynieria Systemów Informatycznych

Data urodzenia: 24 kwietnia 1991 r.

Data rozpoczęcia studiów: 1 października 2010 r.

Życiorys

Nazywam się Marcin M. Urodziłem się w Warszawie w 1991 r. W latach 2007–2010 uczęszczalem do XXVII Liceum Ogólnokształcącego im. Tadeusza Czackiego w Warszawie do klasy o profilu matematyczno–informatycznym, którą zakończyłem egzaminem maturalnym. W październiku 2010 r. rozpocząłem studia na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Od 2013 r. zawodowo zajmuje się przeglądarkami internetowymi.

. . . . . .

podpis studenta

Egzamin dyplomowy

Złożył egzamin dyplomowy w dniu . . . . . .

z wynikiem . . . . . .

Ogólny wynik studiów . . . . . .

Dodatkowe wnioski i uwagi Komisji . . . . . .

. . . . . .

Streszczenie

Niniejsza praca przedstawia zagadnienie klasyfikacji w kontekście ekstrakcji informacji ze stron WWW. Opisuje ich strukturę oraz problemy automatycznego wydobywania danych. Prezentuje także autorski system przetwarzania stron wykorzystujący mechanizm rozszerzeń przeglądarki Chrome. Pokazane rozwiązanie używa atrybutów wizualnych takich jak rozmiar, kolor czy pozycja, aby nauczyć algorytmy klasyfikacji rozpoznawania elementów, naśladując sposób w jaki robią to ludzie.

Słowa kluczowe: *klasyfikacja, ekstrakcja informacji, strony internetowe, SVM, Naiwny Klasyfikator Bayesa, uczenie maszynowe.*

Abstract

Title: *Lists classification on the Web pages using visual features.*

This thesis presents the classification problem in context of extracting information from Internet sites. It describes the structure of the Web pages and issues in automated data mining. It also shows author's system for the Web pages processing, which utilizes Chrome extension mechanism. Outlined solution uses visual features such as size, color or position to train classification algorithms recognize elements simulating the way humans do.

Key words: *classification, information extraction, Web pages, SVM, Naive Bayes Classifier, machine learning*

Spis treści

1. Wstęp

Witryny w sieci WWW zawierają ogrom informacji, których z każdym dniem jest coraz więcej. Ludzie powoli przestają być w stanie samodzielnie przetworzyć taką ilość danych. Dlatego to zadanie coraz częściej zleca się komputerom.

W roku 2013 hakerzy, podszywając się pod agencję *Associated Press*, umieścili fałszywą informację o eksplozjach w *Białym Domu*. W ciągu następnych trzech minut wartość indeksu amerykańskiej giełdy spadła o 136 miliardów dolarów. *Bloomberg* ocenia, że tak duży spadek był wynikiem działania algorytmów przetwarzających wiadomości prasowe [1].

Usługa *Google Now* zaskakuje użytkowników spersonalizowanymi wiadomościami takimi jak: numer terminalu, z którego odlatuje nasz samolot, lista zabytków znajdujących się w miejscu naszego pobytu, wyniki meczu drużyny sportowej, którą się interesujemy. Prezentacja tych treści nie byłaby możliwa bez automatycznego przetwarzania i klasyfikacji fragmentów stron internetowych.

Jak pokazują powyższe przykłady, dziedzina eksploracji danych ma bardzo szerokie zastosowania. Dynamicznie się rozwijając, oferuje rosnącą liczbę udogodnień w zarządzaniu informacjami. Z tego powodu jej fragment stał się materiałem do niniejszej pracy.

W następnym rozdziale opisuję ogólne zagadnienie klasyfikacji, wraz ze sposobem działania kilku algorytmów. Rozdział trzeci poświęcony jest podstawom struktury dokumentów HTML oraz przykładom list demonstrowanych w formie graficznej. W rozdziale czwartym zawarty jest opis architektury systemu klasyfikującego. Rozdział piąty przedstawia wybrane szczegóły implementacyjne. Rozdział szósty prezentuje uzyskane wyniki oraz sposób doboru uwzględnionych atrybutów.

1.1. Definicje używanych terminów

Poniżej przedstawiam opis podstawowych terminów, określający ich znaczenie w niniejszej pracy.

— **Lista** to fragment strony internetowej, który zawiera elementy o podobnej strukturze, pełniący funkcję wyliczenia niezależnych obiektów o charakterze informacyjnym. Roli listy nie pełnią elementy nawigacyjne. Przykłady list na stronach to: zbiór wiadomości, wyniki meczów piłkarskich, rezultaty wyszukiwania.

— **Atrybut, cecha** to wartość lub zbiór wartości liczbowych, opisujących właściwości elementu strony internetowej, reprezentowanego przez węzeł w jej drzewie DOM.

1.2. Cel pracy

Celem pracy jest stworzenie i przetestowanie systemu potrafiącego znaleźć listy na stronach internetowych. Podstawowe założenia to:

— wykorzystanie algorytmów klasyfikacji,

— wykonanie podsystemu do ręcznego oznaczania list,

— użycie przeglądarki internetowej do pobierania danych opisujących elementy stron.

2. Zagadnienie klasyfikacji

Słownik języka polskiego PWN [2] opisuje klasyfikację jako podział osób, przedmiotów, zjawisk na grupy według określonej zasady. W informatyce zagadnienie to jest jedną z metod eksploracji danych, czyli przetwarzaniu przez komputer określonych informacji, w celu odkrycia ukrytych w nich prawidłowości. Przykładem może być określenie czy klientka sklepu internetowego spodziewa się dziecka (przypisanie do grupy kobiet w ciąży) tylko i wyłącznie na podstawie analizy przeglądanych przez nią produktów. Sklep, posiadając taką wiedzę, może zasugerować kupno odpowiednich towarów ze swojej oferty.

Należy odróżnić klasyfikację od regresji, używanej w statystyce. Koncentruje się ona na problemach numerycznych – jako wynik przewidywań otrzymujemy wartości liczbowe, w klasyfikacji natomiast są to etykiety (klasy).

W tej pracy skupiam się na klasyfikatorach binarnych, wykorzystujących uczenie maszynowe z nadzorem. Słowo binarny w powyższym wyrażeniu oznacza podział obiektów na dwie grupy. Uczenie maszynowe to dziedzina sztucznej inteligencji zakładająca stworzenie systemu potrafiącego doskonalić się przy pomocy dostarczanych danych. Budowa takiego systemu składa się z dwóch etapów:

— uczenia klasyfikatora wykorzystując zgromadzone próbki danych reprezentowane przez wektory ich cech,

— właściwej klasyfikacji danych wejściowych.

Nadzór w procesie uczenia polega na tym, że każda próbka na wejściu ma przypisaną klasę. W przypadku klasyfikacji binarnej będzie to wartość boolowska. Alternatywą jest uczenie bez nadzoru, w którym klasyfikator, na podstawie próbek, samodzielnie określa przynależność do klas, wyszukując anomalie.

Kluczowe przy trenowaniu klasyfikatora są odpowiednio dobrane dane uczące. Liczba próbek nie może być zbyt mała. Powinny one równomiernie odzwierciedlać rozkład klas oraz wzorców cech w spodziewanych danych. W przeciwnym razie może dojść do przeuczenia, czyli zbyt dużego dopasowania się klasyfikatora do danych trenujących. Warto jednak zauważyć, że od pewnego momentu zwiększanie ilości próbek ma coraz mniejsze znaczenie.

2.1. Naiwny Klasyfikator Bayesa

Naiwny Klasyfikator Bayesa opera się na modelu probabilistycznym. Korzysta on z twierdzenia Bayesa:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{2.1}$$

gdzie H oznacza hipotezę, że próbka danych X należy do określonej klasy. P(H | X) jest poszukiwaną podczas klasyfikacji wartością, pozwala ona mając próbkę danych przypisać ją do jednej z klas (wybieramy tą o największym prawdopodobieństwie warunkowym). P(H) oznacza prawdopodobieństwo, że dowolna próbka należy do określonej klasy – gdy zatem dodamy do siebie te wartości dla każdej z występujących klas otrzymamy wartość 1. Analogicznie P(X | H) to prawdopodobieństwo wystąpienia konretnej próbki gdy znamy tylko klasę do jakies jest ona przypisana.

Przykład: niech H definiuje, że jakaś osoba posiada w domu biały kapelusz, X to zaobserwowanie na ulicy człowieka w białej marynarce:

— P(H | X) wyraża szansę, że widziana przez nas osoba posiada biały kapelusz,

— P(X | H) szansę posiadania przez kogoś białej marynarki gdy wiemy, że ma on biały kapelusz,

— P(H) oznacza odsetek ludzi chodzących po ulicy i posiadających białe kapelusze,

— P(X) szanse zobaczenia na ulicy kogoś w białej marynarce.

Przydatność powyższego twierdzenia wynika z faktu, że podczas procesu nauki jesteśmy w stanie wyliczyczyć prawdopodobieństwa stojące po prawej stronie równania.

Podczas klasyfikacji zwykle nie interesuje nas jednak konkretne prawdopodobieństwo, a jedynie klasa, która odpowiada za jego maksymalną wartość. Zatem w równaniu 2.1 możemy pominąć P(X), gdyż jest to taka sama wartość dla każdej klasy.

Prawdopodobieńswa P(H) możemy szacować jako liczbę próbek trenujących przypisanych do danej klasy podzieloną przez liczbę klas. Alternatywnie ten czynnik można pominąć gdy nie jesteśmy go w stanie ustalić z dobrym przybliżeniem.

W praktyce obliczenie wszystkich wartości P(X | C) nie jest możliwe ponieważ przykładowo dla 4 atrybutów przyjmujących 100 różnych wartości mamy 100^4 możliwych kombinacji. Dlatego przyjmuje się *naiwne* założenie o niezależności atrybutów. To znaczy, że konkretna wartość jednego z nich nie wpływa na wartość innego atrybutu. Zatem można zastosować poniższy wzór:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \tag{2.2}$$

gdzie k to numer porządkowy atrybutu z próbki X, x_k to jego wartość, a C_i jedna z klas.

Poszczególne $P(x_k|C_i)$ dla każdej z klas C_i obliczymy w następujący sposób:

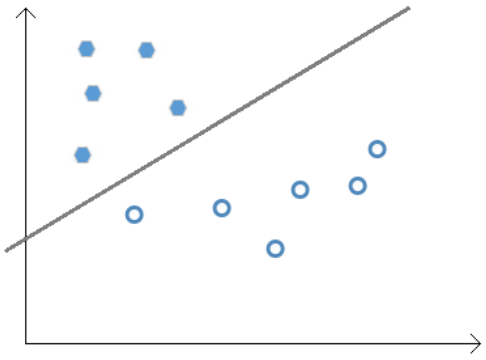
— jeżeli atrybut jest dyskretny to prawdopodobieństwo będzie liczbą próbek trenujących przypisanych do danej klasy i posiadających wartość x_k atrybutu k,

— gdy atrybut jest typu ciągłego należy przyjąć jakiś rozkład prawdopodobieństwa, w książce [3] jako standardowy podaje się rozkład Gaussa. Następnie możemy określić parametry danego rozkładu obliczając wartości takie jak średnia i odchylenie standardowe.

2.2. Klasyfikator SVM

Klasyfikator SVM, nazywany także metodą wektorów nośnych, jest jednym z najpopularniejszych klasyfikatorów. Jak podaje [3] wynika to z jego dużej dokładności oraz większej niż w innych metodach odporności na przetrenowanie.

Najprostszym przypadkiem dla SVM będzie dwuwymiarowa przestrzeń z próbkami należącymi do dwóch klas A i B, ułożonymi w ten sposób, że da się oddzielić prostą, te rezprezentujące tylko jedną z nich. Ilustruje to poniższy rysunek:



Rysunek 2.1: Przykład próbek liniowo rozdzielnych. Opracowano na podstawie [3].

Dowolna prosta wybrana w ten sposób umożliwi nam klasyfikację, jednak aby zapewnić jak najmniejszy poziom błędów SVM stosuje strategię prostej (w ogólnym przypadku: hiperpłaszczyzny) z maksymalnymi marginesami. Formalnie hiperpłaszczyznę

definiujemy:

$$W * X + b = 0 \tag{2.3}$$

gdzie W to wektor współczynników, b to skalarna wartość przesunięcia, X to wektor określający położenie punktu w przestrzeni. Przy danej hiperpłaszczyźnie dla każdej próbki możemy policzyć odległość euklidesową pomiędzy nimi. Margines to suma odległości jednej próbki z klasy A i jednej i klasy B podzielona przez 2. Gdy ta wartość jest najmniejsza, wtedy próbki są zapisywane i tworzą wektory nośne. Faktyczna hiperpłaszczyzna dzieląca nie jest zapisywana i nie jest to ta sama dla której obliczyliśmy początkowe odległości, znajduje się ona w odległości równej marginesowi od obu wektorów nośnych.

W praktyce nie dokonuje się przedstawionych powyżej obliczeń, a sprowadza problem znelezienia minimalnych marginesów do kwadratowego wypukłego problemu optymalizacji, lecz to zagadnienie nie będzie tutaj opisywane.

Co jednak gdy próbki nie są liniowo rozdzielne? Stosujemy wtedy ich transformację do wyższego wymiaru, używając przekształcenia nieliniowego. Takie przekształcenie definiowane jest za pomocą funkcji jądrowej.

2.3. Kryteria oceny

Aby poprawnie ocenić jakość klasyfikatora należy zdefiniować odpowiednie wskaźniki. Poniżej przedstawiam najczęściej stosowane miary dla klasyfikacji binarnej:

$$\textit{dokładność} = \frac{TP + TN}{P + N} \tag{2.4}$$

$$\textit{czułość} = \frac{TP}{P} \tag{2.5}$$

$$\textit{specyficzność} = \frac{TN}{N} \tag{2.6}$$

$$\textit{precyzja} = \frac{TP}{TP + FP} \tag{2.7}$$

$$F_\beta = \frac{(1 + \beta^2) \times \textit{precyzja} \times \textit{czułość}}{\beta^2 \times \textit{precyzja} + \textit{czułość}} \tag{2.8}$$

TP – liczba poprawnie sklasyfikowanych próbek klasy A (pozytywnej),

TN – liczba poprawnie sklasyfikowanych próbek klasy B (negatywnej),

P – liczba wszystkich próbek klasy A,

N – liczba wszystkich próbek klasy B,

FP – liczba próbek klasy B błędnie sklasyfikowanych jako próbki klasy A (błąd pierwszego rodzaju),

β – waga określająca ile razy bardziej cenimy czułość niż precyzję.

Dokładność to odsetek poprawnie odgadniętych klas dla wszystkich próbek. Czułość wskazuje odsetek znalezionych próbek klasy A. Specyficzność jest analogiczna do czułości, ale dotyczy klasy B. Precyzja to odsetek prawidłowo przypisanych klas dla próbek sklasyfikowanych jako należące do klasy A.

Walidacja krzyżowa

Mierzenie jakości klasyfikacji dla próbek wcześniej użytych do wytrenowania klasyfikatora nie odzwierciedla dobrze rzeczywistych warunków. Otrzymane wyniki będą w takim przypadku zbyt optymistyczne. Standardowo algorytmy pracują na danych nieznanych w trakcie uczenia. Aby symulować taką sytuację stosuje się techniki walidacji krzyżowej. Polega ona na podziale zbioru próbek na grupy i przeprowadzaniu uczenia oraz testów na osobnych podzbiorach.

Użyta w niniejszej pracy walidacja jest wariacją walidacji k-krotnej. Zbiór treningowy dzielony jest na kilka grup, następnie każda z nich wykorzystywana jest do nauczenia jednego klasyfikatora. Metryki jakości wyliczane są przy użyciu reszty grup. Następnie liczona jest średnia arytmetyczna metryk dla wszystkich wygenerowanych klasyfikatorów. W ten sposób można zarówno wykorzystać wszystkie posiadane próbki do nauki, jak i zachować odpowiednie proporcje pomiędzy liczbą danych uczących, a liczbą danych testowych.

3. Budowa stron internetowych

Obecnie witryny internetowe do prezentowania treści użytkownikowi stosują najczęściej połączenia trzech technologi: hipertekstowego języka znaczników (HTML), kaskadowych arkuszy stylów (CSS) oraz języka skryptowego JavaScript.

Struktura strony wraz z jej treścią definiowana jest za pomocą znaczników HTML, które zagnieżdżone tworzą układ drzewiasty nazywany obiektowym modelem dokumentu (DOM). Do manipulowania właściwościami wizualnymi służy CSS. Dzięki niemu można ustalić typografię, kolorystykę, odstępy jak i ogólny układ strony. Jednak, w celu uczynienia witryny interaktywną, musimy zastosować JavaScript, umożliwia on manipulację drzewem DOM oraz atybutami CSS, pozwala ustalić reakcję na dynamiczne zdarzenia takie jak ruchy myszą, naciśnięcia klawiszy, załadowanie dokumentu. Wykorzystuje się go także do pobierania danych bez przeładowywania strony.

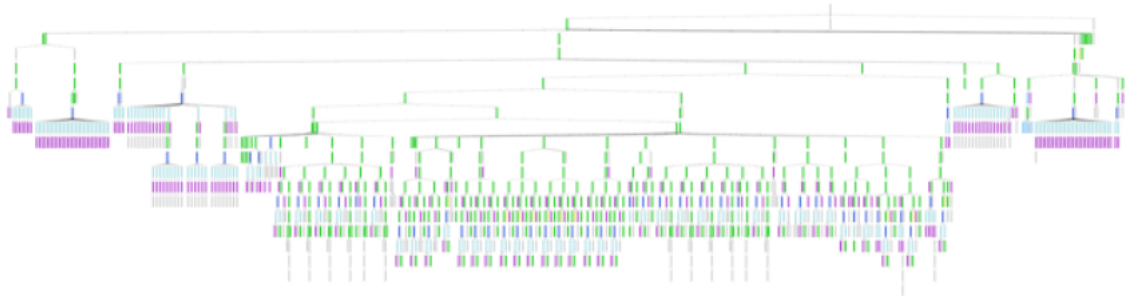
Opisane powyżej technologie nie powstały jednocześnie. W roku 1991 zaprezentowany został HTML jako część usługi WWW, stworzonej przez Tima Berners-Lee. Następnie w roku 1994 zaproponowano CSS, aby odseparować formę dokumentów od ich treści. Jednak do tego czasu HTML posiadał już częściową obsługę modyfikacji wyglądu poprzez znaczniki takie jak <FONT>, <BLINK> czy <CENTER>, niektóre z nich są używane do dziś. Jako ostatni dodano w roku 1995 język JavaScript. Oczywiście od tamtego czasu technologie te nieustannie ewoluują, zachowując wsteczną kompatybilność, co powoduje, że przetwarzanie stron internetowych jest bardziej skomplikowane, niż mogłoby się wydawać.

Teoretycznie niektóre znaczniki posiadają przypisany typ informacji jaki powinny zawierać, tak jak <LI> określa element listy, <P> paragraf, <FOOTER> stopkę, w praktyce jednak nie jest to rygorystycznie przestrzegane. Przykładowo, lista może być przedstawiona jako tabela z jedną kolumną, czy zbiór elementów <DIV> definiujących sekcje.

Kilka zamieszczonych poniżej obrazów demonstruje złożoność stron internetowych na przykładzie witryny *bbc.com*. Rysunek 3.2 przedstawia warstwy, jakie tworzą zagnieżdżone elementy strony, podczas gdy rysunek 3.3 pokazuje to samo jako strukturę drzewiastą. Prezentowana strona zawiera 1200 elementów.

Rysunek 3.1: Przykład strony informacyjnej (http://www.bbc.com/news).

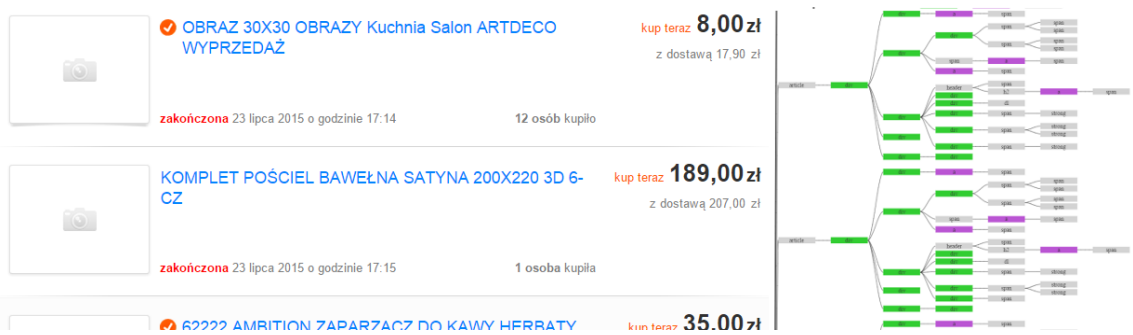
Rysunek 3.2: Ilustracja zagnieżdżenia elementów strony. Wykonano przy pomocy narzędzia Mozilla Tilt [4].



Rysunek 3.3: Ilustracja węzłów drzewa DOM strony. Wygenerowano przy użyciu narzędzia autora pracy.

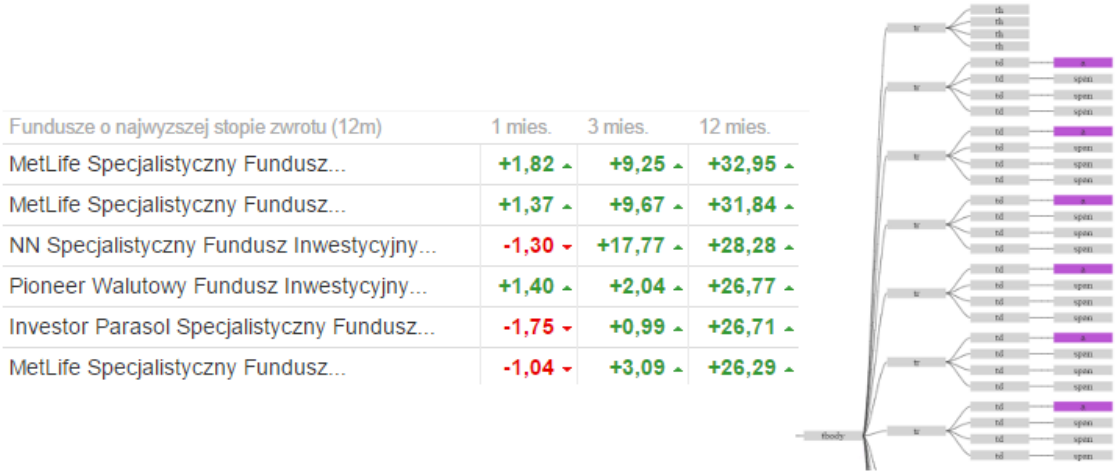
3.1. Przykłady list

W tym miejscu zamieszczam przykłady list zgodnych z definicją z podrozdziału 1.1. Pozwolą one zorientować się Czytelnikowi co, w praktyce, ma rozpoznawać system zbudowany w ramach niniejszej pracy. Poniższe rysunki zawierają wycinki list ze zbioru uczącego wraz z odpowiadającymi im fragmentami drzewa DOM. Widać, że powtarzająca się struktura elementów listy ma swoje odzwierciedlenie w regularnościach drzewa.



Rysunek 3.4: Fragment listy przedmiotów oferowanych na akukcji wraz z odpowiadającym jej drzewem DOM. Źródło: http://allegro.pl.

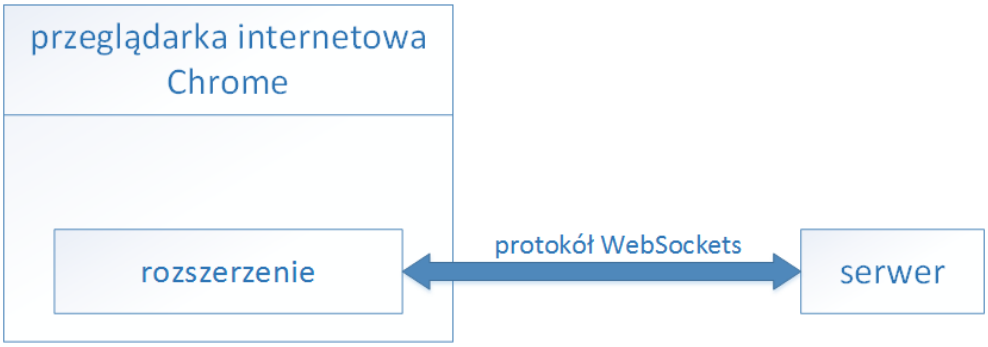
Rysunek 3.5: Fragment listy godzinowej prognozy pogody wraz z odpowiadającym jej drzewem DOM. Źródło: http://pogoda.interia.pl.



Rysunek 3.6: Fragment listy funduszy emerytalnych wraz z odpowiadającym jej drzewem DOM. Źródło: http://www.money.pl/gielda/.

4. Architektura systemu

System klasyfikacji stworzony w ramach niniejszej pracy składa się z dwóch głównych części: rozszerzenia przeglądarki internetowej oraz modułu serwerowego, w którym umieszczono algorytmy klasyfikatorów. Ten zgrubny schemat ilustruje rysunek 4.1. Taki podział został ustalony z kilku powodów, najważniejsze z nich opisuję w poniższych akapitach.



Rysunek 4.1: Podstawowy schemat architektury.

Realizacja poszczególnych zadań jest dużo łatwiejsza za pomocą środowisk do tego dostosowanych. Przeglądarki internetowe świetnie radzą sobie w przetwarzaniem stron internetowych – to ich główne zadanie. Możemy zatem pozwolić przeglądarce pobierać, parsować, renderować strony i od niej uzyskiwać interesujące nas informacje. Niektórzy decydują się na własne rozwiązania obsługujące język HTML oraz CSS, tak jak autorzy pracy [5]. Jednak oni sami stwierdzają, że parsowanie HTML sprawiło im trudności dlatego, że strony internetowe często nie są zgodne ze standardami. Z racji

skomplikowania specyfikacji musieli także zrezygnować z obsługi styli CSS, które służą do elastycznego modyfikowania wyglądu elementów stron. Praca została opublikowana w roku 2002, gdy Internet był mniej rozwinięty, a konstrukcja stron prostsza. Za pomocą takiego rozwiązania uzyskalibyśmy dziś znacznie gorsze rezultaty. Jednak osadzanie popularnych bibliotek klasyfikatorów bezpośrednio w rozszerzeniu jest utrudnione bądź niemożliwe (zależnie od przeglądarki) z tego powodu zostały one umieszczone w części serwerowej.

Zastosowanie architektury klient-serwer implementuje model oprogramowania jako usługi. Wiele nowych komercyjnych systemów jest realizowanych w ten sposób. Jako przykład można podać usługi Google takie jak: tłumacz, synteza mowy czy rozpoznawanie tekstu. Dzięki temu uzyskujemy większą elastyczność, a zmiany w klasyfikatorach łatwiej wdrażać. Można ulepszać algorytmy pozyskując dodatkowe dane od klientów.

4.1. Rozszerzenie przeglądarki internetowej

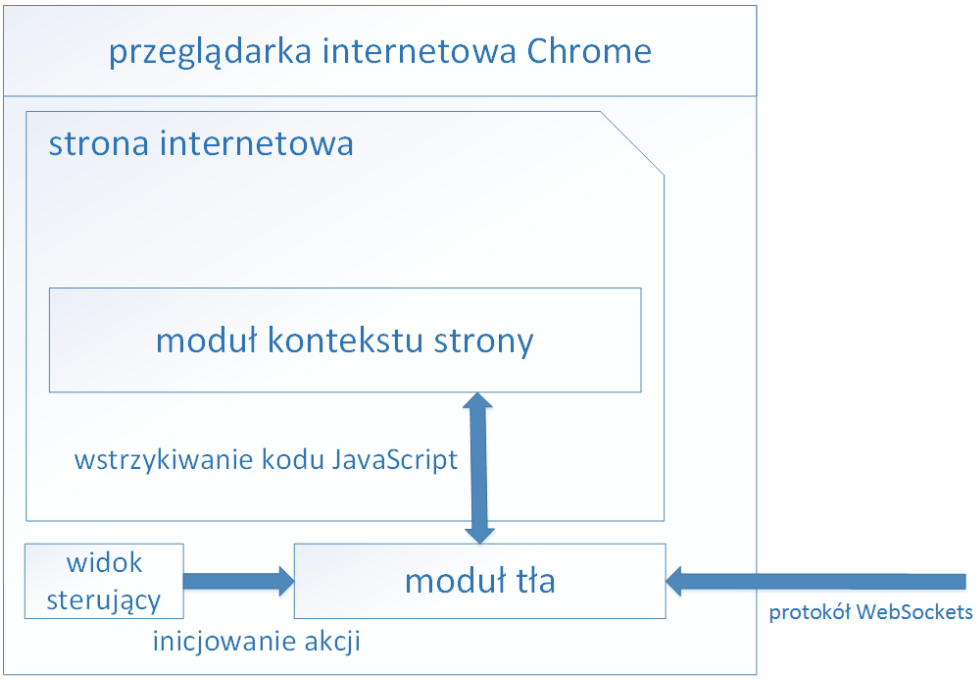
Rozszerzenie napisane jest w języku JavaScript z minimalistycznym widokiem sterującym w HTML i CSS. Wykorzystuje ono mechanizm rozszerzeń Chrome, z którym kompatybilna jest także Opera oraz Microsoft Edge. Składa się ono z dwóch głównych fragmentów:

— moduł tła,

— moduł kontekstu strony,

— widok sterujący.

Każde działanie jest rozpoczynane z poziomu widoku sterującego. Po wybraniu akcji przekazywana jest wiadomość do modułu tła. Następnie zestawiane jest połączenie



Rysunek 4.2: Schemat architektury rozszerzenia.

z serwerem. Używam protokołu WebSockets z powodu natywnego wsparcia w przeglądarce. Przykładowe akcje to: rozpoczęcie manualnego oznaczania stron, prezentacja testowych stron wraz z markerami, test klasyfikacji na aktywnej stronie.

Moduł tła zajmuje się przygotowaniem strony oraz serwera do wykonania zleconej akcji. Przesyła wiadomość inicjującą do serwera. Jeżeli akcja dotyczy aktywnej strony, wtedy wstrzykuje on moduł kontekstu strony. Następnie oczekuje na nadejście z serwera rozkazów wykonania funkcji takich pobranie wartości atrybutów, selekcja losowego elementu czy oznaczenie elementu prostokątem w celu prezentacji wyników.

Moduł kontekstu strony zajmuje się operacjami na drzewie DOM dokumentu do którego został wstrzyknięty. Przeglądarka internetowa pozwala na dostęp do treści strony wyłącznie skryptom umieszczonym w ten sposób. Używając API JavaScript możemy pobierać ze strony informacje takie jak: rozmiar i kolor tekstu, pozycję i rozmiar

elementów oraz ich typ. Mamy także dostęp do drzewa DOM, za pomocą którego enumerujemy wszystkie elementy strony.

Nietypową operacją zaimplementowaną w module kontekstu strony jest pobranie danych treningowych. Uruchamia ona prosty interfejs pozwalający na oznaczanie elementów strony jako list.

Komunikacja pomiędzy widokiem sterującym, a modułem tła odbywa się przy pomocy przekazywania wiadomości opisanego w dokumentacji [6]. Wstrzykiwanie kodu do strony oraz odbiór wyników używają mechanizmu zaprezentowanego w [7]. Protokół WebSocets, niezbędny do połączenia z serwerem, przedstawiono natomiast na stronie [8].

4.2. Serwer

Serwer został napisany w języku C++. Jego szkielet oparty jest na bibliotece POCO [9], która umożliwia implementację serwera WebSockets. Aplikacja składa się z następujących części:

— serwer WebSockets,

— kontroler akcji,

— moduł zdalnego wywoływania procedur (RPC),

— menadżer uczenia,

— zarządca archiwów Warc,

— klasyfikatory.

Większość z wymienionych części opakowana jest w osobne klasy, dodatkowo serwer posiada szereg pomocniczych funkcji do serializacji próbek oraz obsługi formatu JSON, używanego przy RPC.

Rysunek 4.3: Schemat modułów serwera.

Po starcie serwer nasłuchuje na porcie TCP. Standardowo cała komunikacja odbywa się w ramach pojedynczego połączenia, które posiada niezależny kontroler akcji. Protokół WebSockets oparty jest na TCP, ale jego nietypową cechą jest inicjowanie połączenia za pomocą HTTP. W początkowej fazie wysyłane jest żądanie HTTP z użyciem nagłówka Upgrade, dopiero później następuje właściwa komunikacja. Szczegółowe informacje na temat protokołu można znaleźć pod adresem [8].

Kontroler akcji jest bardzo minimalistyczny, otrzymuje on ciąg znaków reprezentujący nazwę inicjowanej akcji oraz ewentualne argumenty oddzielone znakiem spacji. W obrębie kontrolera tworzony jest moduł zdalnego wywoływania procedur oraz menadżer uczenia.

Moduł zdalnego wywoływania procedur zawiera wszystkie funkcje wykonywane w środowisku przeglądarki internetowej, takie jak: otwarcie konkretnej strony, pobranie ze strony informacji treningowych, enumeracja elementów na stronie, pobranie położenia elementów.

Po rozpoczęciu akcji treningu menadżer uczenia wykorzystuje zarządce archiwów Warc oraz moduł zdalnego wywoływania procedur, aby zlecać ładowanie kolejnych

stron internetowych. Następnie pobiera on dane treningowe, które wprowadza użytkownik. Gdy wszystkie strony są już oznaczone przez użytkownika uruchamia on poprzez kontroler akcji operację zapisu danych do plików json.

Zarządca archiwów Warc jest odpowiedzialny za sterowanie zewnętrzym programem, który ładuje z dysku archiwa Warc zawierające zapisane grupy stron internetowych. Szegółowy opis użytego programu znajduje się w sekcji 5.4.

Klasyfikatory posiadają opcję trenowania oraz dynamicznego tagowania elementów dla zadanej strony. Dostęp do nich odbywa się za pośrednictwem kontrolera akcji. Procedury trenowania wykorzystują dane zebrane przez menadżera uczenia, a zapisane w postaci plików json. Pliki te, jako mechanizm pośredniczący, zostały wykorzystane aby uniknąć wielokrotnego otwierania stron podczas testowania różnych parametrów klasyfikatorów, gdyż jest to bardzo czasochłonne.

5. Implementacja

W poniższym rozdziale prezentuję szegóły implementacji klasyfikatorów i dokładnie opisuję atrybuty, których przydatność będę weryfikował w rozdziale 6. Przedstawiam także systemy zdalnego wywoływania procedur oraz przechowywania testowego zbioru stron, ponieważ są one zrealizowane w ciekawy sposób.

5.1. Klasyfikacja

Strony składają się z węzłów drzewa DOM. Problem klasyfikacji list można sformułować jako przypisanie każdego węzła do jednej z dwóch klasy. Do pierwszej gdy reprezentuje, wraz ze wszystkimi elementami potomnymi, jakąś listę, bądź do drugiej w przeciwnym wypadku. Jest to zatem klasyfikacja binarna.

Klasyfikatory zostały napisane w języku C++, są one umieszczone w części serwerowej. Kod SVM pochodzi z biblioteki dlib [10], implementacja oparta jest na algorytmie *Sequential Minimal Optimization*. Jest on parametryzowany przy pomocy wartości C i γ, które należy dobrać eksperymentalnie. Wydruk 5.1 przedstawia procedurę ich poszukiwania. Zastosowałem metodę siatki 30 na 30 punktów w skali logarytmicznej dla zakresów C od 0.001 do 100000 i γ od 0.000001 do 100. Kod Naiwnego Klasyfikatora Bayesa został oparty na implementacji zamieszczonej w serwisie GitHub [11], stosuje on rozkład Gaussa przy przetwarzaniu atrybutów.

```
typedef matrix<double, 3, 1> sample_type;  // Trzy atrybuty.
typedef radial_basis_kernel<sample_type> kernel_type;
std::vector<sample_type> samples;
std::vector<double> labels;
// Dla czytelności pominięto kod ładowania próbek.
randomize_samples(samples, labels);
svm_c_trainer<kernel_type> trainer;
double best_gamma = 0, best_C = 0, best_f2 = 0;

matrix<double> params = cartesian_product(
  logspace(log10(1e-6), log10(100), 30), // gamma
  logspace(log10(100000), log10(1e-3), 30) // C
);

for (long col = 0; col < params.nc(); ++col)
{
    const double gamma = params(0, col);
    const double C = params(1, col);

    trainer.set_kernel(kernel_type(gamma));
    trainer.set_c(C);

    double f2  = cross_validation(trainer, samples, labels,
      10); // Walidacja 10-krotna.

    if (f2 > best_f2) {
      best_f2 = f2;
      best_gamma = gamma;
      best_C = C;
    }
}
```

Wydruk 5.1: Dobór parametrów klasyfikatora SVM przy użyciu biblioteki Dlib.

5.2. Atrybuty

Do klasyfikacji niezbędna była implementacja cech, którymi można byłoby opisać próbki. Posługując się własnym wyczuciem, określiłem następujące atrybuty:

— rozmiar i położenie,

— rozkład koloru tekstu,

— rozkład wielkości tekstu,

— wewnętrzne podobieństwo struktury (pq-Gram),

— średnia liczba słów wśród elementów potomnych,

— minimalna wariancja rozmiaru i położenia.

Implementacja atrybutów została wykonana w języku JavaScript, znajduje się ona w części rozszerzenia przeglądarki.

Aby określić rozmiar i położenie każdemu elementowi strony przypisano okalający go prostokąt i pobrano jego współrzędne oraz wielkość podzielone przez wielkość strony. Taka normalizacja została przeprowadzona, aby zminimalizować wpływ rozdzielczości ekranu na wyniki. Motywacją dla tego atrybutu było założenie, że listy będą znajdować się w typowych miejscach na stronie, przykładowo najczęściej w jej środku.

Rozkład koloru tekstu ma za zadzanie pokazać czy listy mają tendencję do zawierania jaśniejszych, bądź ciemniejszych kolorów niż reszta elementów. Aby go uzyskać sumowane są wszystkie piksele z jakich składa się tekst zawarty, niekoniecznie bezpośrednio, w danym elemencie. Ze względów praktycznych brana jest pod uwagę wartość koloru w skali szarości, czyli średnia arytmetyczna z jego składowych r, g, b. Wydruk 5.2 prezentuje wykorzystaną procedurę zliczania pikseli tekstu. Jest ona bardziej skomplikowana niż zliczanie liter, ale może zostać rozszerzona o obsługę rysunków. Implementując ją, chciałem pokazać duże możliwości języka JavaScript.

```
function countTextPixels(node) {
  var style = window.getComputedStyle(node);
  var font = style.font;
  var fontSize = parseInt(style.fontSize);
  var text = getNodeText(node);

  if (!text.trim())
    return 0;

  var canvas = document.createElement('canvas');
  var context = canvas.getContext("2d");
  context.font = font;
  canvas.width = context.measureText(text).width;
  canvas.height = fontSize * 2;
  context.font = font;
  context.fillText(text, 0, fontSize);

  var pixelCount = 0;
  var imageData = context.getImageData(0, 0,
    canvas.width, canvas.height);
  var data = imageData.data;
  for (var i=0; i < data.length; i += 4) {
    //var r = data[i];
    //var g = data[i + 1];
    //var b = data[i + 2];
    var alpha = data[i + 3];
    if(alpha > 0)
      ++pixelCount;
  }

  return pixelCount;
}
```

Wydruk 5.2: Zliczanie pikseli tekstu w języku JavaScript.

Każdemu elementowi został również przypisany rozkład rozmiaru czcionki w przeliczeniu na liczbę znaków napisanych tym rozmiarem. Tekst w elementach potomnych był uwzględniany.

Do określenia wewnętrznego podobieństwa struktury została użyta miara pq-Gram opisana w [12]. Jest to odpowiednik tekstowej odległości edycyjnej dla struktur drzewiastych. W implementacji wykorzystałem bibliotekę [13]. Wartość atrybutu pq-Gram dla elementu została określona jako średnia arytmetyczna sumy miary podobieństwa pq-Gram dla każdej pary sąsiadujących, bezpośrednich potomków elementu (jego dzieci).

Zaimplementowanie obliczania średniej liczby słów wśród bezpośrednich elementów potomnych (dzieci) obiektu wynika z przypuszczenia, że listy powinny zawierać kilka słów na każdy element. Liczba słów była wyliczana z uwzględnieniem elementów potomnych.

Minimalna wariancja rozmiaru i położenia zależy od czterech wartości: pionowego i poziomego położenia oraz wysokości i szerokości. Liczona jest ich wariancja wsród dzieci węzła, a następnie brana pod uwagę najmniejsza wartość. Elementy listy prawie zawsze mają jakiś wspólny układ na stronie, przykładowo, są ustawione jeden pod drugim lub są podobnych rozmiarów. Pozytywną własnością tego atrybutu jest jego duża niezależność od struktury DOM, zakłada on tylko, że elementy listy mają wspólny korzeń. Ten atrybut powinien dobrze się uzupełniać z atrybutem pq-Gram, który bada wyłącznie strukturę DOM.

5.3. Zdalne wywoływanie procedur

Aby móc połączyć generowanie atrybutów w przeglądarce internetowej oraz implementację klasyfikatorów w formie serwera potrzebny był pomost w postaci RPC. Do tego celu wykorzstałem protokół WebSockets oraz format JSON. Powodem była ich natywna obsługa w przeglądarce.

```
class RemoteProcedureCall {
public:
    RemoteProcedureCall(const string& function_name,
        int pageId);
    void AddArgument(string name, Json::Value value);

    string GetCallString() const;
    bool ParseResult(const string& result);
    const Json::Value& GetResult() const;

private:
    Json::Value json_in_;
    Json::Value json_out_;
};

class RPCMachine;

double RPCMachine::PqGramDistanceChildrenMean(int nodeUid,
    int pageId) {
  RemoteProcedureCall rpc("PqGramDistanceChildrenMean",
      pageId);
  rpc.AddArgument("uid", nodeUid);
  Call(rpc);
  return rpc.GetResult().asDouble();
}

void RPCMachine::Call(RemoteProcedureCall& rpc) {
  io_->Write(rpc.GetCallString());
  scoped_ptr<string> result = io_->Read();
  rpc.ParseResult(*result);
}
```

Wydruk 5.3: Kod zdalnego wywołania procedury na przykładzie pobrania atrybutu pq-Gram.

Na wydruku 5.3 zamieściłem fragment implementacji RPC. Klasa *RemoteProcedureCall* opakowuje serializację i deserializację zdalnego wywołania. Każda procedura

posiada nazwę oraz identyfikator strony, do której jest kierowana. Funkcje globalne, takie jak otwarcie nowej strony, posiadają specjalny identyfikator. Klasa *RPCMachine* zawiera definicje wszystkich zdalnych procedur w formie funkcji składowych. Pod koniec prac nad serwerem, w celu zwiększenia wydajności, została dodana zdalna procedura, pobierająca w jednym wywołaniu atrybuty wszystkich elementów strony.

5.4. Przechowywanie testowego zbioru stron

Najbardziej popularne witryny, odwiedzane przez miliony osób, budowane w celach komercyjnych, zawierają szybko zmieniające się treści. Nie jest możliwe pobieranie ich za każdym razem, gdy potrzeba próbek w kolejnej iteracji algorytmów klasyfikujących ponieważ otrzymalibyśmy, dla tej samej strony, różniące się od siebie dane. Należy zatem zbudować podsystem odpowiedzialny za przechowywanie egzemplarzy stron lokalnie. Takie zadanie w prezentowanym systemie pełni zarządca archiwów Warc wraz z programem WebArchivePlayer [14].

Najpierw strony WWW zostały zapisane przy pomocy usługi internetowej WebRecorder.io [15] do formatu Web ARChive (w skrócie Warc), opisanego standardem ISO 28500:2009. Został on zaprojektowany do tego, aby jak najwierniej i najefektywniej przechowywać każdy typ danych przesyłany protokołem HTTP. Nadaje się on także do zastosowań profesjonalych, jest używany przez crawler Heritrix, napisany na potrzeby cyfrowej biblioteki Internet Archive.

Następnie, w celu odtworzenia stron, uruchamiany jest program WebArchivePlayer [14], który serwuje je poprzez lokalny serwer HTTP. Z perspektywy przeglądarki załadowanie takiej strony nie różni się od wczytania jej z internetu.

Niestety praktyka pokazała, iż powyższe programy nie radzą sobie perfekcyjnie z dynamicznie ładowanymi treściami. Zdolność do ponownego dopasowania oznaczonych wczesniej 250 elementów ze zbioru 16 popularnych w polsce witryn wynosi około 88%. Również wydajność lokalnego serwera programu WebArchivePlayer [14] nie jest zbyt wysoka. Może być to spowodowane użyciem w implemetacji języka Python, który w pewnych zastosowaniach jest bardzo mało wydajny. Dla wspomnianego wczesniej zbioru 250 próbek sumaryczny czas ich wczytywania wynosi 6 minut. Ostatnią, ale najbardziej dotkliwą, wadą są okresowe błędy w odtwarzaniu strony. Czasem strona wygląda zupełnie inaczej niż powinna, gdyż nie zostały załadowane kluczowe zasoby takie jak plik z arkuszami stylów. Źródło tego problemu nie zostało zidentyfikowane, ale błędy tego typu pojawiają się dość rzadko, mniej więcej 1 raz na 90 ładowań.

6. Eksperymenty

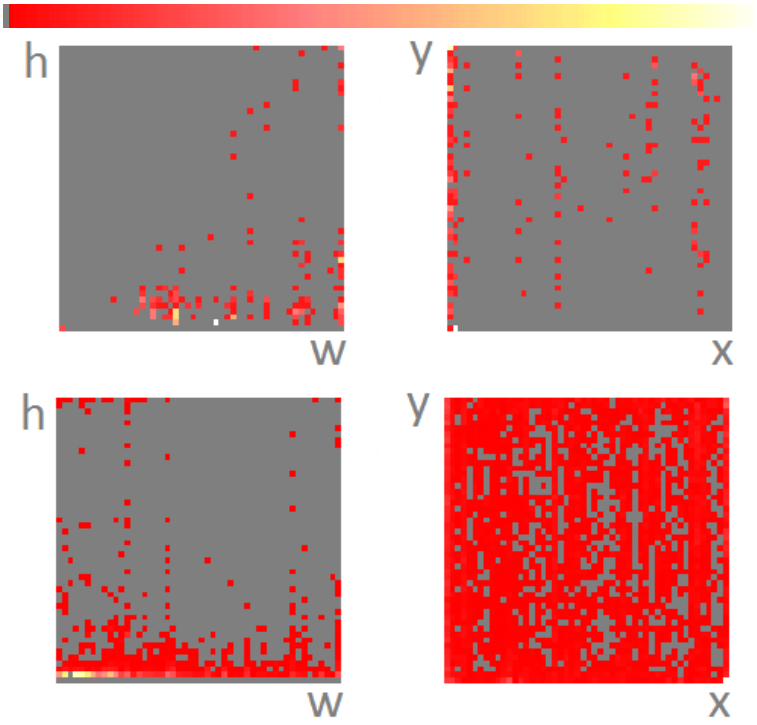
Bardzo ważnym etapem jest dostrojenie wybranych algorytmów i danych trenujących tak, aby uzyskać jak najwyższą jakość klasyfikacji. W tym celu należy dobrać odpowiednie atrybuty oraz ocenić metryki jakości klasyfikacji dla różnych parametrów implementacyjnych.

Duże znaczenie ma także zbiór danych testowych, nie może być on zbyt mały i powinien odzwierciedlać realne dane. Jako próbek trenujących i testowych użyłem zbioru 250 ręcznie oznaczonych list oraz 5200 losowych elementów zebranych z 16 spośród 25 najpopularniejszych w Polsce witryn według rankingu Alexa [16] w czerwcu 2015 roku. Przy konstrukcji jednego z histogramów zastosowałem większą liczbę 13000 próbek losowych.

6.1. Dobór atrybutów

W tej sekcji przedstawiam analizę atrybutów zaimplementowanych w ramach niniejszej pracy.

Jako pierwsze zostały przetestowane wartości położenia i rozmiaru obiektów. Jak pokazuje rysunek 6.1 korelacja tych atrybutów z klasą próbki występuje, lecz jest zbyt mała by być użyteczną w praktyce. Na żółto oznaczono pola odpowiadające większej liczbie elementów, na czerwono te mniejszej liczbie, a na szaro brak elementów.



Rysunek 6.1: Dwuwymiarowe histogramy atrybutów rozmiaru i położenia dla 230 próbek list oraz 13000 próbek losowych.

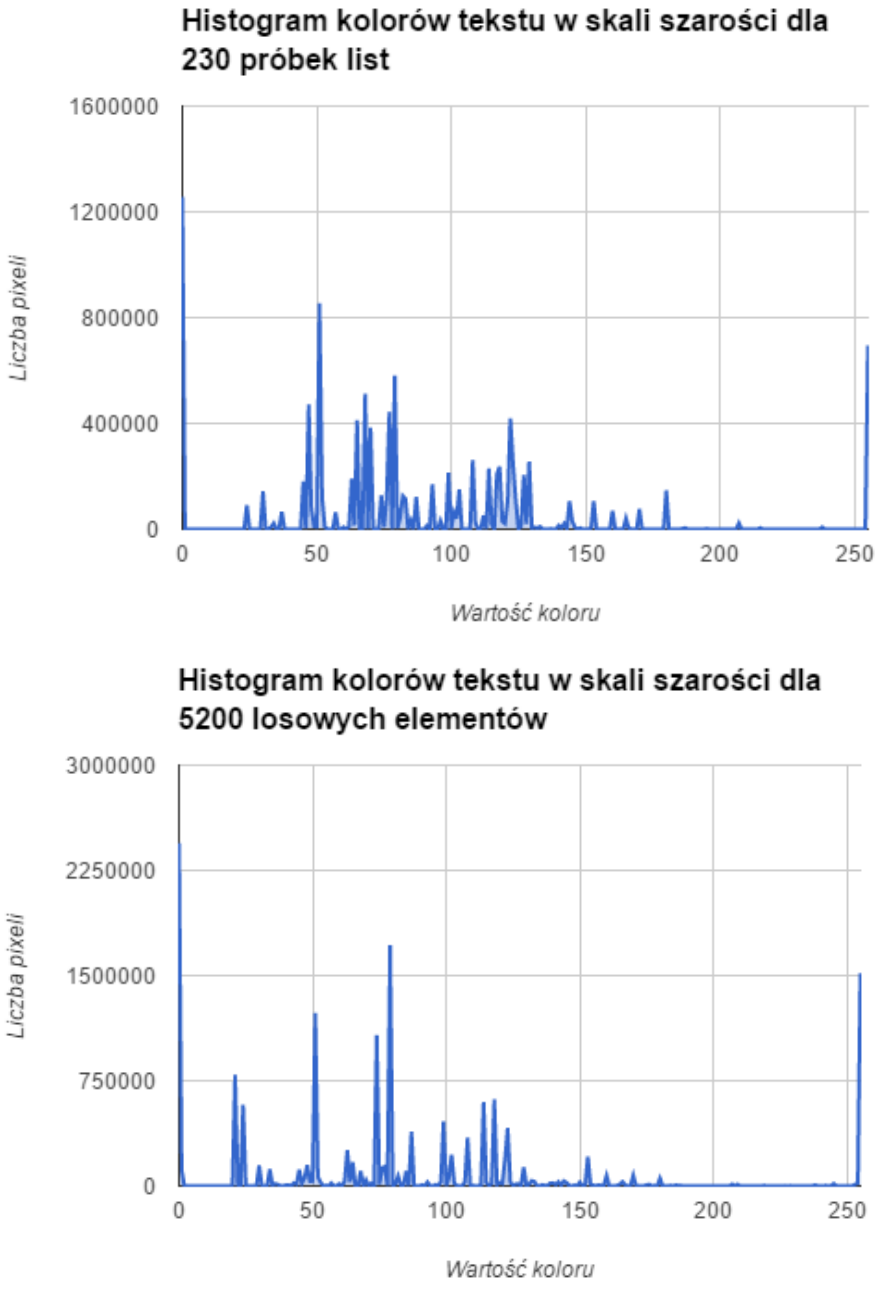
Następnie przeanalizowałem kolory tekstu. Na rysunku 6.2 można zaobserwować całkowity brak korelacji.

Podobnie mało przydatny jak poprzedni atrybut okazał się rozmiar tekstu. Rysunek 6.3 przedstawia te rozkłady zsumowane dla wszystkich branych pod uwagę próbek. Sprawdziłem również rozkłady znormalizowane, zliczające udział procentowy znaków, a nie ich liczbę, lecz histogramy były bardzo podobne.

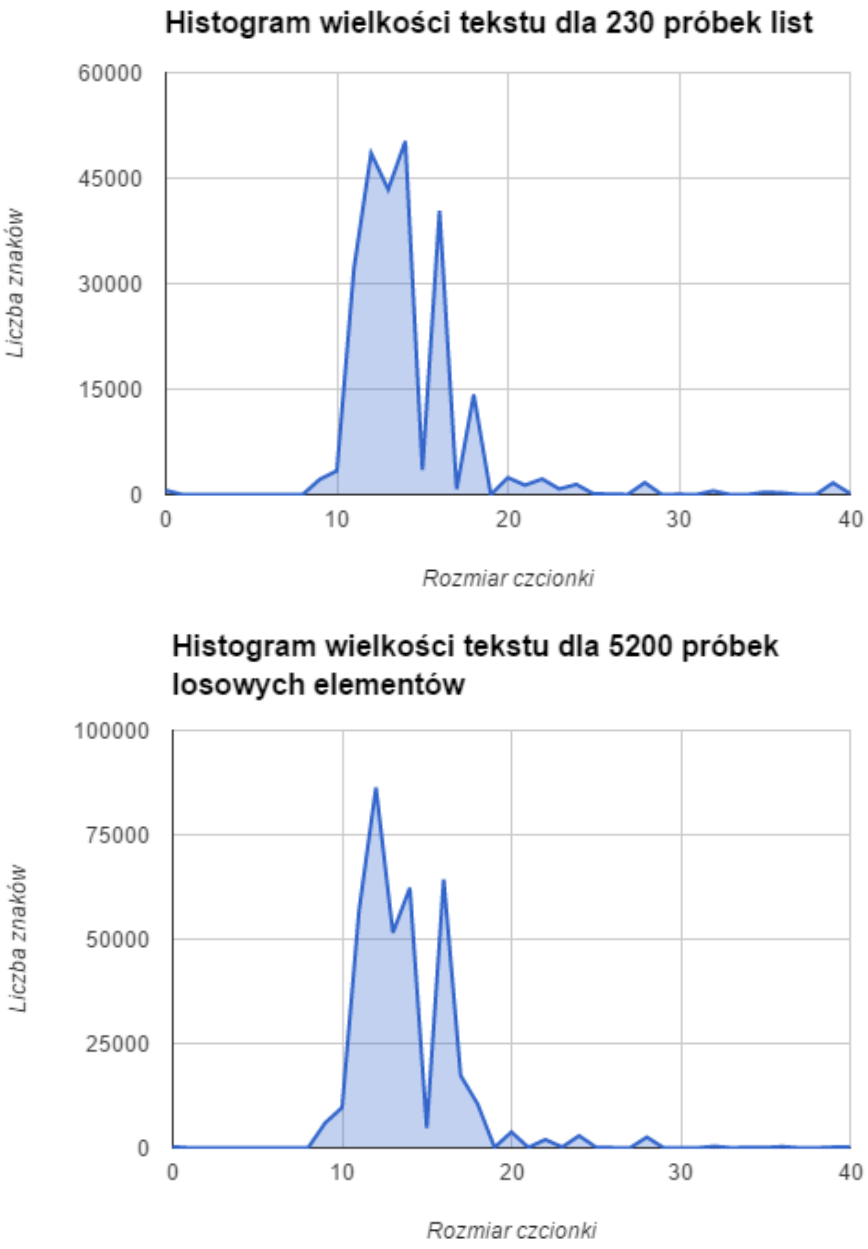
Najbardziej obiecującym okazał się atrybut wewnętrznego podobieństwa struktury. Rysunek 6.4 pokazuje bardzo istotną korelację, dla list wartości są bliskie zeru, co oznacza duże podobieństwo, natomiast dla próbek losowych najczęściej pojawia się wartość jeden, odzwierciedlająca brak podobieństwa.

Przeanalizowałem także średnią liczbę słów wśród bezpośrednich elementów potomnych (dzieci) obiektu. Na wykresach z rysunku 6.5 widać pewną zależność. Losowe elementy mają najczęściej od zera do czterech słów na dziecko, a listy od trzech do trzydziestu. Zatem większe wartości tego atrybutu są dobrym predyktorem występowania listy.

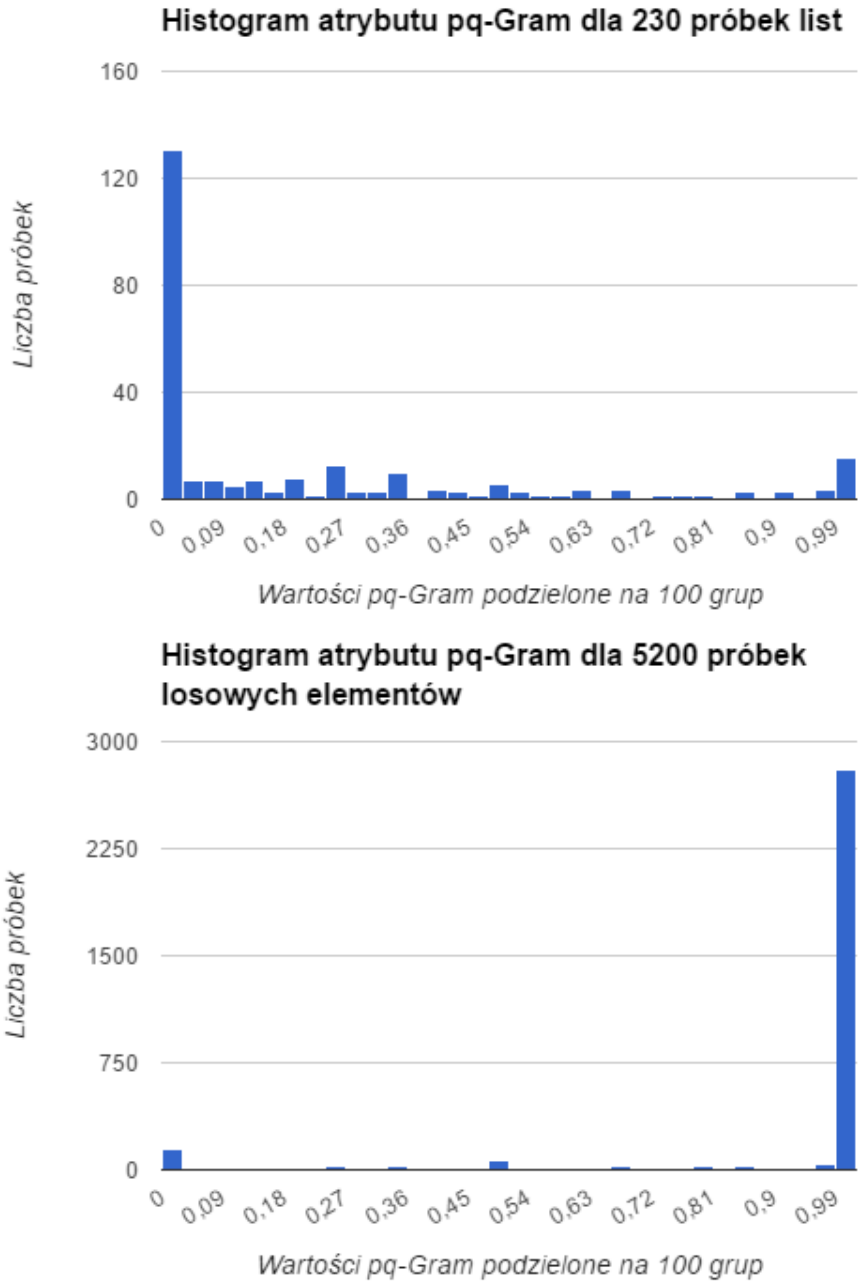
Zaskakująco dobrym atrybutem jest minimalna wariancja rozmiaru i położenia. Wartości dla zbioru list oraz przypadkowych elementów są porównane na rysunku 6.6.



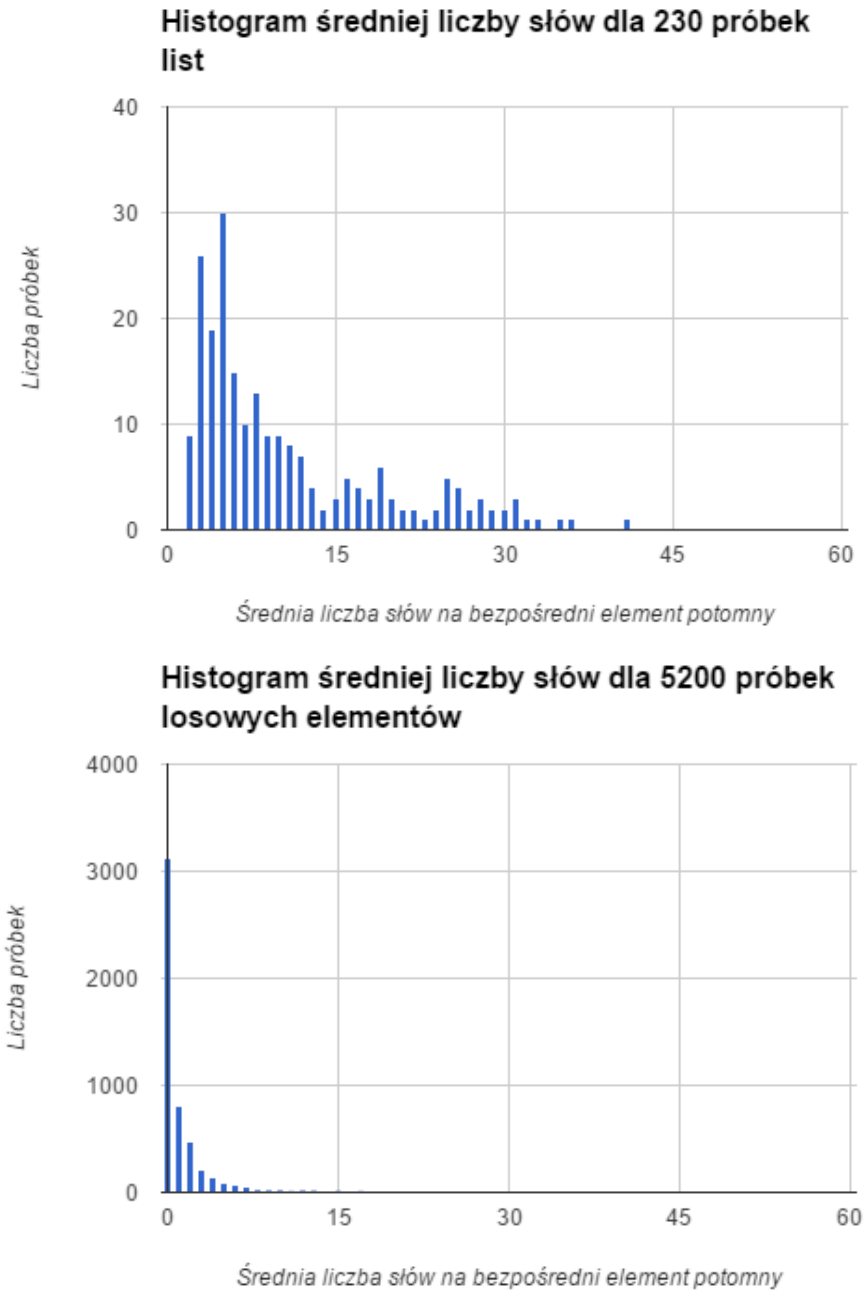
Rysunek 6.2: Wykresy rozkładów koloru tekstu.



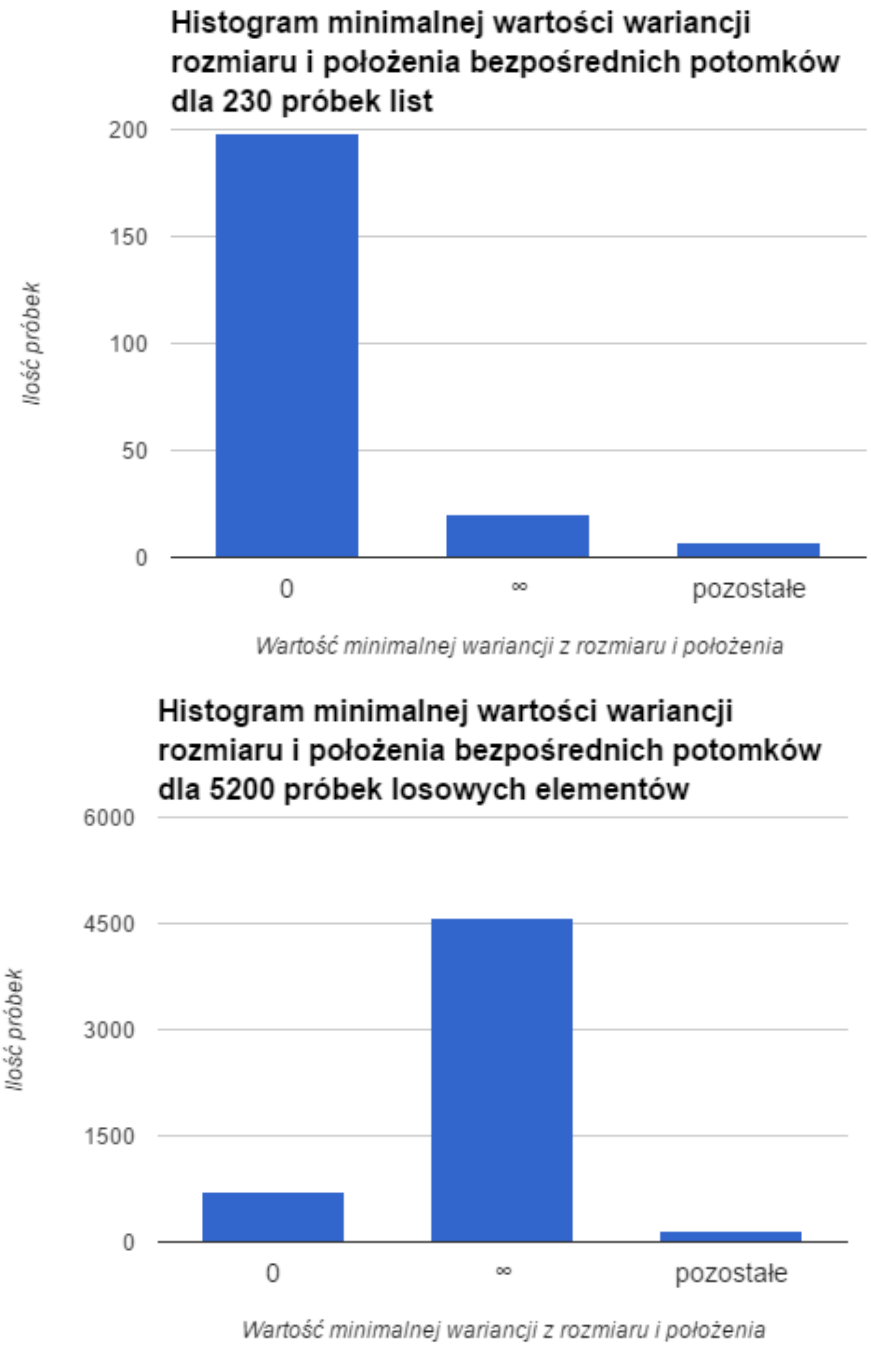
Rysunek 6.3: Wykresy rozkładów wielkości tekstu.



Rysunek 6.4: Wykresy wewnętrznego podobieństwa struktury.



Rysunek 6.5: Wykresy średniej liczby słów wśród elementów potomnych.



Rysunek 6.6: Wykresy minimalnej wariancji rozmiaru i położenia.

6.2. Wyniki

Ten podrozdział prezentuje skuteczność klasyfikacji, według kryteriów z podrozdziału 2.3, dla różnej konfiguracji parametrów i atrybutów. Ostatecznie jako atrybuty zastosowano: pq-Gram, średnią liczbę słów wśród elementów potomnych oraz minimalną wariancję rozmiaru i położenia.

Jako funkcji jądrowej klasyfikatora SVM użyto RBF, natomiast Naiwny Klasyfikator Bayesa stosuje rozkład Gaussa. Przetestowano również funkcję sigmoidalną dla SVM oraz rozkłady Bernoulliego i wielomianowy, jednak uzyskane wyniki były znacząco niższe, dlatego zdecydowałem się ich nie przedstawiać.

Rezultaty walidacji krzyżowej dla zbioru 250 próbek list oraz 250 próbek negatywnych (nie będących listami) pokazuje tabela 6.1. Wadą takiego sposobu oceny jest stosowanie takich samych proporcji próbek obu klas do uczenia i ewaluacji. Jest to niepożądane, ponieważ w rzeczywistości występuje duża dysproporcja pomiędzy przykładami pozytywnymi i negatywnymi.

Walidacja krzyżowa k-krotna	k = 2		k = 10	
	SVM	Naive Bayes	SVM	Naive Bayes
Dokładność [%]	89	90	86	90
Czułość [%]	98	93	98	90
Precyzja [%]	79	87	76	88
F_2 [%]	94	91	93	90

Tabela 6.1: Wyniki walidacji krzyżowej.

Aby uzyskać wyniki bliższe rzeczywistej skuteczności klasyfikatorów użyłem różnych podzbiorów grupy testowej do trenowania i ewaluacji. Jednocześnie przetestowałem wpływ proporcji próbek pozytywnych i negatywnych podczas uczenia na skuteczność klasyfikacji. Widać, że proporcja ta określa kompromis pomiędzy czułością,

a precyzją. Zależność przedstawiono w tabeli 6.2. Wartości parametrów oceniających obliczono dla 250 próbek pozytywnych i 5200 próbek negatywnych, co odzwierciedla mały odsetek list wśród wszystkich elementów realnych stron, w zbiorze testowym wynosił on około 0,03%.

Próbki grupy trenującej, X pozytywnych + Y negatywnych	SVM		Naive Bayes	
	Czułość [%]	Precyzja [%]	Czułość [%]	Precyzja [%]
125 + 1716	52	40	85	35
125 + 125	96	20	89	29
125 + 0	100	4	100	4

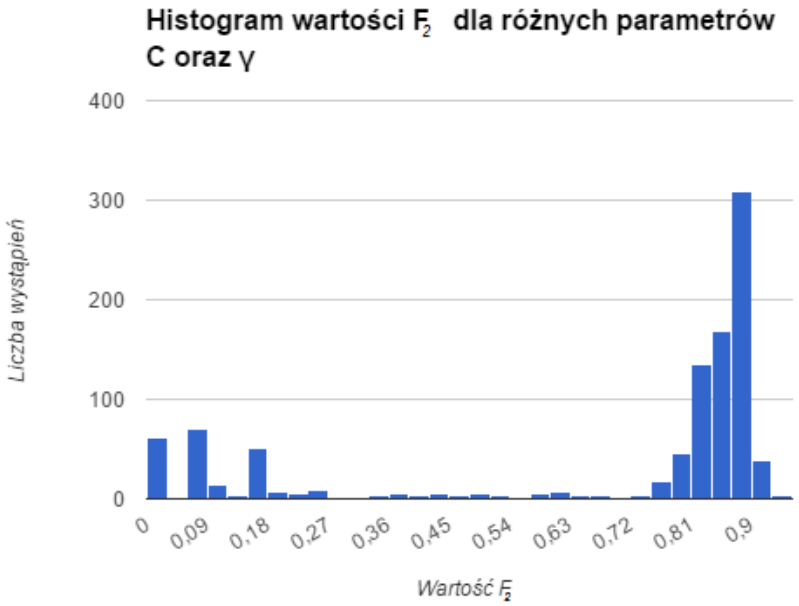
Tabela 6.2: Ocena klasyfikacji trenowanej z różnym udziałem próbek negatywnych, wyliczona dla 250 próbek pozytywnych i 5200 próbek negatywnych.

		F_2 [%]	
		SVM	Naive Bayes
Jeden atrybut	a) pq-Gram	89	85
	b) średnia słów	85	78
	c) minimalna wariancja	88	89
Dwa atrybuty	a) + b)	91	85
	a) + c)	88	88
	b) + c)	89	89
Trzy atrybuty	a) + b) + c)	93	90

Tabela 6.3: Wyniki walidacji krzyżowej 10-krotnej w zależności od zastosowanych atrybutów.

W tabeli 6.3 pokazuję, że stosowanie trzech atrybutów daje większą skuteczność niż użycie każdego z nich osobno, bądź parami. Jest zatem uzasadnione użycie kilku atrybutów do trenowania klasyfikatorów. Dodatkowo, wybór każdego z zastosowanych atrybutów był dobrą decyzją, gdyż zwiększył jakość klasyfikacji.

Rysunek 6.7 przedstawia histogram uzyskanych miar F_2 dla różnych wartości parametrów klasyfikatora SVM. Niektóre z nich dają znacznie niższe wyniki, ale większość skutkuje uzyskaniem zbliżonych rezultatów. Jednak, nawet dla poprawienia skuteczności o 10%, warto zastosować przeszukiwanie przestrzeni tych parametrów.



Rysunek 6.7: Wykres oceny klasyfikacji wyrażonej miarą F_2 dla różnych wartości parametrów C i γ klasyfikatora SVM.

7. Podsumowanie

W ramach niniejszej pracy udało się stworzyć system rozpoznający listy na stronach internetowych. Uzyskałem 96% czułość w odnajdywaniu list przy 20% precyzji. Odróżnianie elementów podobnych do list, ale nimi niebędących, takich jak menu czy stopka okazało się trudne. Najprawdopodobniej mała precyzja wynika z braku filtrowania danych wejściowych, nie dokonano segmentacji elementów. Bardzo często listy są wykrywane wewnątrz innych list, co nie powinno być dozwolone i może zostać odfiltrowane z użyciem dodatkowych technik.

Uwzględnienie atrybutów wizualnych (rozmiaru i położenia) pozwoliło zwiększyć skuteczność klasyfikatora SVM o około 2%, a Naiwnego Klasyfikatora Bayesa o 5%. Jednocześnie powinno poprawić działanie dla nietypowych przypadków, ponieważ zmniejsza wrażliwość klasyfikatorów na anomalie w strukturze DOM strony.

System wykorzystujący przeglądarkę internetową daje dużo większe możliwości poszukiwania atrybutów. W łatwy sposób pozwala je pozyskiwać, jak i łączyć ze strukturą dokumentu. Nieprzetworzone atrybuty takie jak położenie elementu są mało przydane, jednak uwzględnienie struktury dokumentu i obliczenie, przykładowo, wariancji wartości dla elementów potomnych pozwala poprawić klasyfikację.

Bibliografia

[1] Lu Wang, Whitney Kisling, Eric Lam. Fake post erasing 136 billion shows markets need humans. http://www.bloomberg.com/news/articles/2013-04-23/fake-report-erasing-136-billion-shows-market-s-fragility. Data dostępu: 13.08.2015.

[2] Słownik języka polskiego PWN. http://sjp.pwn.pl. Data dostępu: 15.07.2015.

[3] Jiawei Han, Micheline Kamber, Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, wydanie trzecie, 2011.

[4] Mozilla Tilt: Visualize your web page in 3d. https://hacks.mozilla.org/2011/07/tilt-visualize-your-web-page-in-3d/. Data dostępu: 06.09.2015.

[5] Miloš Kovaeevia, Michelangelo Diligenti, Marco Gori, Marco Maggini, Veljko Milutinovia. Web page classification using visual information. *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, strony 250–257, 2002.

[6] Chrome extensions message passing. https://developer.chrome.com/extensions/messaging. Data dostępu: 05.08.2015.

[7] Chrome extensions Content Scripts programmatic injection. https://developer.chrome.com/extensions/content_scripts#pi. Data dostępu: 05.08.2015.

[8] WebSockets protocol. https://www.websocket.org/aboutwebsocket.html. Data dostępu: 05.08.2015.

[9] POCO C++ libraries. http://pocoproject.org/index.html. Data dostępu: 05.08.2015.

[10] Biblioteka Dlib. http://dlib.net/. Data dostępu: 06.09.2015.

[11] Implementacja Naiwnego Klasyfikatora Bayesa. https://github.com/timnugent/naive-bayes. Data dostępu: 06.09.2015.

[12] Nikolaus Augsten, Michael Böhlen, Johann Gamper. Approximate matching of hierarchical data using pq-grams. *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, strony 301–312. VLDB Endowment, 2005.

[13] JQGram tree edit distance approximation, Javascript port of PyGram with some additional functionality. https://github.com/hoonto/jqgram. Data dostępu: 16.08.2015.

[14] WebArchivePlayer. https://github.com/ikreymer/webarchiveplayer. Data dostępu: 16.08.2015.

[15] WebRecorder. https://webrecorder.io/. Data dostępu: 16.08.2015.

[16] Ranking popularności stron Alexa. http://www.alexa.com/topsites/countries/PL. Data dostępu: 20.06.2015.