Revista Română de Interacţiune Om-Calculator 4(2) 2011, 131-156 © MatrixRom

WikiDetect: Sistem de detecţie automată a vandalismului pe Wikipedia

Dan Cioiu, Traian Eugen Rebedea

Universitatea Politehnica din Bucureşti – Facultatea de Automatică şi Calculatoare
Splaiul Independenţei, Nr. 313, 060042, Sector 6, Bucureşti
E-mail: dan.cioiu@gmail.com, traian.rebedea@cs.pub.ro

Rezumat. Vandalismul a fost mereu una dintre cele mai mari probleme ale Wikipedia, însă există puţine soluţii automate de rezolvare a acesteia. Voluntarii petrec mult timp corectând articolele vandalizate, timp care ar putea fi folosit pentru a îmbunătăţi Wikipedia în alte moduri. Scopul acestei lucrări este de a propune un sistem nou de detecţie a vandalismului bazat pe învăţare automată de la un corpus adnotat de articole vandalizate şi de a-i analiza perfomanţele. Mediul de lucru al acestei aplicaţii este unul foarte apropiat de realitate, întrucât lucrează pornind de la wikitext extras direct din baza de date a enciclopediei i adnotat de către experţi, care este folosit pentru evaluarea standard a aplicaţiilor pentru detecţia vandalismului pe Wikipedia. În ultima parte a lucrării, vom realiza o analiză critică a performanţelor obţinute, făcând referire la soluţii deja existente şi vom încerca să maximizăm eficienţa algorimului de detecţie folosind diverse metode de clasificare statistică.

Cuvinte cheie: Wikipedia, vandalism, spam, clasificare.

1. Introducere

Wikipedia este o enciclopedie online colaborativă, bazată pe articole, pe care orice persoană cu acces la Internet o poate edita. Modificarea unui articol are în majoritatea cazurilor scopul îmbunatăţirii acestuia prin inserarea de completări sau corectări obiective şi constructive. În multe cazuri, proprietatea de enciclopedie deschisă (în engleză *openness*) a Wikipedia permite editorilor cu intenţii răuvoitoare să modifice în sens negativ un articol sau să introducă unul cu caracter de spam (conţinut nesolicitat sau irelevant).

Orice modificare a unui articol de pe Wikipedia poartă numele de *editare* iar starea unui articol la un moment dat se numeşte *revizie* (în engleză *edit*). Wikipedia implementează mecanisme de păstrare a tuturor informaţiilor

despre revizie şi utilizatorul care l-a realizat, precum şi despre toate stările articolului în timp, ca urmare a editărilor repetate.

Lucrarea de faţă are ca scop realizarea unui sistem software de interacţiune cu Wikipedia, capabil să analizeze părţile componente ale unui articol (în special ale unei revizii) şi să identifice trăsături care ar ajuta la separarea cât mai bună a editărilor rău-intenţionate de cele constructive. Metodele oficiale aplicate în prezent sunt: urmărirea de către utilizatori a articolelor şi anularea manuală a editărilor sau folosirea unor programe specializate automate de detecţie a vandalismului.

Vandalismul pe Wikipedia poate fi definit ca o acţiune *rău intenţionată* a unui utilizator care duce la modificarea stării unui articol pe Wikipedia, având un *impact negativ* asupra articolului şi comunităţii de cititori. Trebuie făcută foarte bine diferenţa între vandalism şi acţiuni bine intenţionate, dar cu impact negativ.

Wikipedia a adoptat un sistem ajutător de prevenire a vandalismului prin introducerea unor sisteme automate numite *boţi* (prescurtare de la roboţi) care răspund la comenzi umane (ale administratorilor) şi anulează edit-uri maliţioase, raportează rezultate obţinute sau detectează intenţii ale utilizatorilor de mascare a adresei IP prin folosirea de proxy-uri (servicii de reţele care maschează accesul la o reţea sau adresa reală a unui calculator).

Problema se poate formula ca o sarcină ce constă în separarea unui set de date de intrare în două categorii: o categorie a editărilor *curate* (sau bine-intenţionate) şi cealaltă a editărilor *vandalizate* (sau rău-intenţionate). Vom defini reguli care să determine cu un grad cât mai înalt de încredere dacă o revizie este rezultatul unui act de vandalism şi să implementăm un sistem ce separă datele într-un timp cât mai scurt şi cu rezultate cât mai precise. Pentru această problemă de separare a datelor se va folosi o soluţie clasică în domeniul mineritului de date şi a învăţării automate, şi anume încadrarea acestora în seturi distincte, în urma aplicării unui *clasificator*.

În decursul ultimilor ani au fost încercate mai multe implementări de soluţii ce rezolvă această problemă. Acestea folosesc diverse metode şi abordări, iar rezultate precum cele obţinute de Itakura şi Clarke (2009) sunt demne de luat în considerare. Pornind de la acestea, ne propunem implementarea şi testarea unei soluţii noi de detecţie automată a vandalismului pe Wikipedia. Aceasta trebuie să prelucreze articole Wikipedia ce fac parte dintr-un set de date oficiale folosite pentru evaluarea aplicaţiilor pentru detecţia vandalismului şi să producă rezultate

cuantificabile comparabile sau mai bune cu cele ale soluțiilor existente în acest moment.

Pentru înțelegerea articolului sunt necesare o serie de noțiuni specifice domeniului. De aceea, în continuare este definit un set de termeni care vor fi folosiți în cadrul lucrării:

- **Meta-date** – informații publice, însă invizibile în mod direct, asociate unui articol; exemple: numele autorului, data editării, comentariul, etc.;
- **wikitext** – format text in care sunt scrise şi salvate în baza de date articolele de pe Wikipedia;
- **edit** sau revizie – starea unui articol la un moment dat; conține atât informația vizibilă pe Wikipedia, cât şi meta-datele corespunzătoare;
- **editare** – acțiunea unui utilizator de a schimba starea unui articol;
- **revenire** – acțiunea de aducere a unui articol într-o stare precedentă;
- **feature** – o caracteristică a unei revizii ce poate fi cuantificată;
- **parsare** – traversarea textului în scopul analizei conținutului său;
- **parser –** instrument folosit pentru parsarea textului;
- **edit curat** – revizie regulamentară, care nu se încadrează în categoria reviziilor vandalizate;

Lucrarea continuă cu patru capitole, având următoarea structură: capitolul 2 oferă o vedere generală a arhitecturii sistemului de detecție, iar în capitolul 3 sunt descrise în detaliu deciziile de implementare a aplicației, precum şi clasificatorii folosiți. Capitolul 4 prezintă rezultatele obținute în urma folosirii aplicației implementate şi stadiile prin care s-a trecut în vederea obținerii unor rezultate cât mai bune. La final, în capitolul 5 sunt precizate direcții de proiectare viitoare şi concluziile activităților de cercetare i evaluare efectuate.

2. Arhitectura sistemului

Problema detecției vandalismului pe Wikipedia se abordează prin implementarea unui sistem software capabil să primească la intrare un set de edit-uri şi să specifice la ieşire care dintre acestea reprezintă un act de vandalism, prin intermediul unui marcaj (în engleză *flag*).

Pentru evaluare este folosită analiza unui set de date prin partiționarea sa în date de antrenament şi date de test. Sistemul de detecție acționează ca un

program de învățare automată care calculează un vector de caracteristici ale fiecărei revizii şi, pe baza datelor de antrenament (despre care se ştie în ce categorie se încadrează), învață corespondența dintre valorile vectorului şi cele două categorii. Această corespondență este stabilită de către un motor de învățare automată capabil să aplice anumiți algoritmi asupra vectorilor şi să specifice (cu un anumit grad de încredere) dacă editul corespunzător este vandalizat sau nu. Asupra datelor de test se măsoară eficiența sistemului prin calcularea unor măsuri statistice standard, cum ar fi precizia sau acoperirea.

În *figura 1* este prezentată schema unei astfel de arhitecturi. Cele *N* revizii sub forma unor fişiere text în format MediaWiki sunt parsate (analizate) şi transformate în vectori a câte *F* membri (unde *F* este numărul de caracteristici). Apoi, sunt aplicate cinci metode sau algoritmi de clasificare distincți în vederea comparării rezultatelor obținute. Clasificatorii creează câte un vector a câte *N* valori binare pe care le setează la 1, dacă revizia corespunzătoare a fost vandalizată şi la 0, în caz contrar.

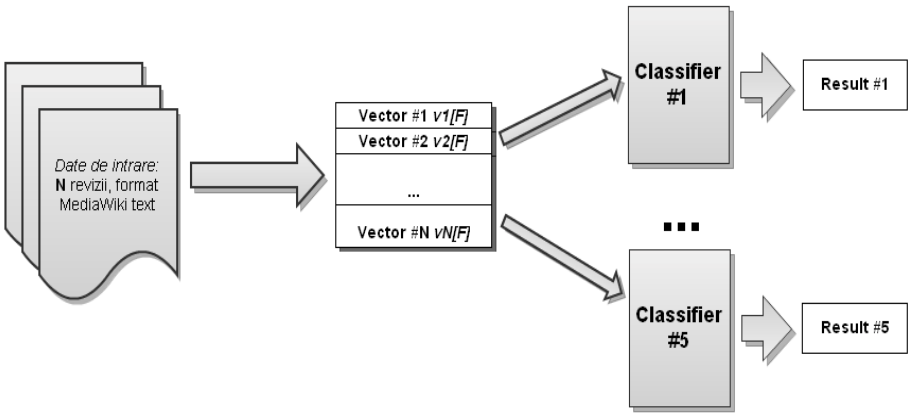

Figura 1. Arhitectura sistemului de detecție a vandalismului pe Wikipedia

Din punct de vedere software, soluția implementată este o aplicație Java ce integrează mai multe tehnologii necesare în diverse stadii de implementare. Acestea sunt: parsarea datelor de intrare, crearea obiectului diferență (în engleză, *diff*) ca o concretizare la nivel obiect a diferențelor dintre două revizii ale aceluiaşi articol, calcularea valorilor din vectorii ce descriu editările, crearea unui fişier ce înglobează vectorii sub un format convenabil, şi încărcarea acestui fişier într-un program capabil să aplice

clasificatorii asupra setului de date. Descrierea tehnologiilor folosite în aceste stadii se va face în subcapitolul 3.3.

Deoarece problema detecției automate a vandalismului pe Wikipedia este una de clasificare binară, ne interesează rezultatele obținute de către cei cinci clasificatori atât în cazul fiecărei clase, cât şi ca medie ponderată între acestea. Metricile de evaluare necesare unei comparații corecte între mai multe seturi de rezultate sunt: procentul de instanțe clasificare corect, rata de observații „*true-positive*" (TP) şi „*false-positive*" (FP), precizia şi acoperirea, scorul F_1 (media armonică între precizie şi acoperire) şi aria de sub curba *ROC*. Desigur, toate aceste metrici vor putea fi calculate după aplicarea clasificatorilor asupra unor date de antrenament, şi vor asigura un anumit grad de încredere metodei de clasificare în momentul aplicării sale pe un set mult mai mare de date de test. Din acest motiv, structura datelor de antrenament şi, respectiv, de test, influențează arhitectura oricărei aplicații de clasificare. Descrierea corpusului de date folosit în aplicația WikiDetect se va face în primul paragraf din capitolul 3.

Datele de intrare sunt de tip fişiere text, iar pentru aplicarea clasificatorilor folosim sistemul Weka (http://www.cs.waikato.ac.nz/ml/weka/). Din acest motiv, aplicația va trebui să primească la intrare fişiere text şi să construiască la ieşire un fişier de tip Attribute-Relation File Format (fişier cu extensia *.arff*).

În funcție de rezultatele analizei performanțelor obținute de către clasificatori, au fost introduse sau eliminate anumite părți ale arhitecturii. În secțiunea de rezultate intermediare vom argumenta adăugarea unui filtru de normalizare a datelor şi a unuia de eliminare a unora din date, precum şi eliminarea parserului MediaWiki sau a clasificatorului Naive Bayes.

3. Implementarea soluției

3.1 Descrierea datelor de intrare

Sistemul WikiDetect foloseşte un corpus de date de dimensiuni mari distribuit de cercetătorii implicați în *International Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse PAN-11* (http://pan.webis.de). Acest corpus coincide cu cel folosit în cadrul conferinței din 2011. Mai multe informații despre corpus sunt disponibile în (Potthast, 2010).

Corpusul este alcătuit dintr-un set de date de antrenament a căror clasă de apartenență a fost stabilită manual sau de către entități autorizate şi un set de date de test a căror clasă nu se cunoaşte, dar care trebuie stabilită. Datele au fost culese din baza de date Wikipedia între 18 noiembrie 2009 şi 6 decembrie 2010, iar apartenența la una dintre cele două clase a fost stabilită pe baza voturilor primite de la utilizatori umani pe Wikipedia.

Datele de antrenament sunt structurate în modul următor: un set de fişiere text reprezentând revizii (conținutul în format wikitext) ale articolelor Wikipedia care sunt fie sursă, fie rezultat al unei operațiuni de editare, un fişier care specifică meta-datele editării, un fişier care specifică clasa fiecărei revizii, şi un fişier care oferă informații despre autorul reviziei.

Datele de test comprimă un număr de peste 300.000 de editări, care trebuie evaluate de către sistemul implementat. Printre acestea se află un număr de 15.000 de editări ale căror clasă de apartenență se cunoaşte doar de către evaluatori şi cu care se va face verificarea eficienței algoritmilor de detecție, precum şi evaluarea performanțelor. La data realizării lucrării de față şi a sistemului software descris, comitetul de program al PAN-11 nu a lansat corpusul de date de test către public, ci doar către participanții la workshop. Din acest motiv, aplicația va fi testată doar pe datele de antrenament, care cuprind peste 32.000 revizii. Dintre acestea, aproape 2.500 au fost considerate de către editorii umani ca fiind vandalism (adică aproximativ 7%, ceea ce verifică rata oficială de pe Wikipedia).

Datele de intrare ale aplicației sunt organizate sub forma unor fişiere:

- **edits.csv** – lista propriu-zisă de revizii; informațiile utile aplicației sunt: id-ul unic al editului, numele utilizatorului (sau adresa IP) care l-a realizat, două id-uri ce fac referire la stările de dinainte şi de după modificare, URL-ul de pe Wikipedia ce prezintă online informația modificării şi comentariul;
- **gold-annotations.csv** – lista ce specifică dacă un edit (menționat prin id-ul său unic) a fost sau nu votat drept act de vandalism pe Wikipedia; sunt menționate numărul de voturi pentru şi, respectiv, împotrivă;
- **annotations.csv** – lista ce specifică pe fiecare linie id-ul unei revizii şi verdictul acordat de către un adnotator (utilizator uman);
- **annotators.csv** – lista cu adnotatorii care au atribuit totalul de voturi exprimate în fişierul precedent.

După cum se observă, votul general acordat în corpus nu determină o corectitudine perfectă, însă, deoarece a fost acordat de către adnotatori umani, gradul de încredere este ridicat - deci detectorul de vandalism se poate antrena folosind aceste date.

Dezavantajul colectării centralizate a editărilor este faptul că ora şi data poartă marca serverului Wikipedia, deci acestea nu pot fi folosite de către aplicație. Aşa cum se arată în (West et al., 2010), este dovedit faptul că informațiile temporale asociate unui edit pot fi folosite cu succes ca o caracteristică a vectorului de trăsături, însă deoarece nu dispunem de ora locală a utilizatorului care acționează, nu putem trage nicio concluzie din acest punct de vedere. De asemenea, considerăm ca fiind irelevante datele despre adnotatori, precum şi ora şi data la care aceştia au acordat voturile.

3.2 Caracteristici analizate

După cum am precizat anterior, un clasificator determină clasa unei instanțe pe baza caracteristicilor sale. Acestea sunt reprezentate sub forma unui vector de valori (reale, binare, nominale, etc.), care, în final, descriu instanța.

În cazul detecției vandalismului pe Wikipedia, au fost luate în calcul o serie de 24 de caracteristici care au fost considerate ca fiind cele mai relevante în procesul de separare a reviziilor corecte de către cele vandalizate. Acestea sunt:

- Max_caractere_consecutive - Cea mai lungă secvență de caractere consecutive;
- Număr_secvențe_caractere_consecutive - Câte secvențe de caractere consecutive prezintă noua stare față de cea precedentă;
- Rata_majusculelor - Numărul de majuscule raportat la numărul total de caractere adăugate;
- Max_cuvânt - Cel mai lung cuvânt al articolului;
- Numărul_pronumelor - Numărul de pronume adăugate;
- Impactul_pronumelor - Variația numărului de pronume după efectuarea reviziei;
- Numărul_cuvintelor_vulgare - Numărul de cuvinte vulgare adăugate;
- Impactul_cuvintelor_vulgare - Variația numărului de cuvinte vulgare după efectuarea reviziei;

- Creşterea_dimensiunii - Mărimea stării curente (în număr de caractere) raportată la mărimea stării precedente (în formă wikitext);
- Asemănarea_între_înlocuiri - Diferența de caractere (în modul) dintre cele două stări consecutive (în format text normal);
- Este_anomin - Specifică dacă editul este anonim sau nu;
- Lungimea_comentariului - Lungimea (în caractere) a comentariului introdus după efectuarea modificărilor;
- Frecvența_cuvintelor_vulgare_comentariu - Numărul de cuvinte vulgare din comentariu raportat la numărul total de cuvinte din comentariu;
- Rata_majusculelor_comentariu - Numărul de majuscule raportat la numărul total de caractere din comentariu;
- Max_cuvânt_comentariu - Cel mai lung cuvânt din comentariul asociat articolului editat;
- Impactul_tagurilor - Variația numărului de etichete mediawiki;
- Impactul_referințelor - Variația numărului de referințe text (de tipul *<ref>*);
- Conține_cuvinte_aleatoare - Valoare binară; se analizează dacă utilizatorul a făcut vreo modificare de un singur cuvânt;
- Impactul_imaginilor - Variația numărului de imagini din articol după efectuarea reviziei;
- Impactul_comentariilor_cod - Numărul de comentarii noi adăugate în codul de editare nou (a nu se confunda cu comentariul reviziei);
- Impactul_cuvintelor_vulgare_comentarii - Numărul de cuvinte vulgare ascunse în comentariile HTML;
- Autor_vandal_istoric - Flag care specifică dacă autorul a vandalizat vreun articol în istoricul său de editări;
- Rata_caractere_nonalpha_comentarii_cod - Numărul de caractere non-alfabetice din comentariile din codul HTML de editare;
- Numărul_cuvintelor_suspecte - Numărul de cuvinte suspecte (dar non-vulgare) adăugate în text, comentariile HTML, sau comentariul reviziei;
- Numărul_inserărilor_identice - Numărul de construcții text identice adăugate la modificare.

3.3 Tehnologii folosite

Aplicația WikiDetect este un program software dezvoltat în limbajul de programare Java care încorporează tehnologii necesare în diverse etape ale implementării, acestea fiind: prelucrarea fişierelor de intrare, parsarea codului wikitext, crearea obiectului *diff* – obiectul ce reprezintă diferențele dintre două revizii, şi prelucrarea şirurilor de caractere.

Biblioteca **Java CSV** (http://sourceforge.net/projects/javacsv/) este o colecție *open-source* de clase Java creată în scopul citirii şi scrierii de text în format CSV („*comma separated values*") şi prelucrării fişierelor de acest tip. Biblioteca aceasta se caracterizează prin dimensiunea mică şi lucrul foarte rapid şi simplu.

În cadrul aplicației noastre, am folosit Java CSV pentru a încărca fişierele CSV din corpusul de date descris anterior. Informațiile sunt citite apoi linie cu linie şi datele necesare sunt extrase.

Fiecare stare a unui articol este prezentă sub forma unui fişier text ce cuprinde conținutul articolului în format wikitext. Pentru a obține textul normal, acesta trebuie parsat cu un instrument de tip parser MediaWiki (formatul în care sunt scrise paginile sursă ale Wikipedia). În aplicația WikiDetect am folosit parserul open-source **Java Wikipedia API - Bliki engine** (http://www.ohloh.net/p/gwtwiki). Mai exact, ne-am folosit de funcțiile care transformă un text de tip wikitext în *plain text* (text normal), dar biblioteca Bliki suportă atât transformări în plain text, cât şi în HTML şi PDF.

Bibliotecile **Diff Match and Patch** (http://code.google.com/p/google-diff-match-patch/) dispun de algoritmi robuşti capabili să prelucreze text, în vederea atingerii scopurilor următoare: crearea obiectului *diff*, găsirea celei mai potrivite potriviri fuzzy (în engleză, *fuzzy match*) dintr-un text (dându-se un şir de căutare), aplicarea unei liste de cuvinte la altele dintr-un text acolo unde există potrivire cu un şir de căutare. Diff Match and Patch oferă implementări în Java, C++, C#, JavaScript, etc., şi folosesc acelaşi API (interfață de programare) şi funcționalități.

Aplicația WikiDetect foloseşte această bibliotecă pentru a construi obiectul *diff*, pe baza a două şiruri de caractere de intrare. Biblioteca implementează algoritmul *Myer's diff* care este considerat cel mai bun în acest scop. De asemenea, algoritmul este îmbunătățit cu o serie de accelerări şi optimizări în vederea obținerii unei calități mai bune şi unui timp scăzut

de operare. Codul prezentat mai jos afişează elementele unei liste de diferenţe de text (şi afişarea lor) între revizia anterioară şi cea curentă:

```
diff_match_patch d = new diff_match_patch();
LinkedList<Diff> l = d.diff_main(oldrevision, newrevision);
for(int i=0; i<l.size(); i++)
     if(l.get(i).operation != diff_match_patch.Operation.EQUAL)
          System.out.println(l.get(i));
```

Clasa `Diff` conţine câmpul *operation* care specifică dacă diferenţa constă în adăugare, ştergere, sau bucata de text este egală cu cea din textul iniţial. Acest lucru este util deoarece analiza acestui câmp oferă posibilitatea extragerii textului adăugat sau şters. Pe baza acestora sunt calculate mai multe din valorile caracteristicilor.

În prelucrarea şirurilor de caractere era nevoie, în unele cazuri, de operaţiuni puternice de analiză a textului. În acest sens, am ales să folosim biblioteca **StringUtils** oferită de Apache (http://commons.apache.org/), care este de fapt o singură clasă ce dispune de un număr mare de metode statice, aplicabile asupra variabilelor de tip şir de caractere. În aplicaţia WikiDetect am folosit StringUtils pentru a transforma toate caracterele ce formează un string în caractere minuscule, dar şi pentru a verifica dacă un şir conţine o secvenţă de caractere.

3.4 Clasificatori folosiţi

În acest paragraf prezentăm aspecte teoretice referitoare la clasificatorii folosiţi în implementări existente, care încearcă să rezolve problema detecţiei vandalismului pe Wikipedia. Algoritmii analizaţi sunt: Support Vector Machines, arbori de decizie (ADTree şi NBTree), Naive Bayes, modelul bazat pe reţele bayesiene şi modelul bazat pe regresii logistice. În afară de aceste metode care au stat la baza cercetărilor descrise în această lucrare, există şi alte abordări ale problemei, de exemplu folosind sisteme bazate pe reguli (Ciucanu et al., 2010).

Învăţarea bazată pe algoritmul **Support Vector Machines** (propus de Vapnik şi Cortes, 1995) face posibilă analiza datelor şi recunoaşterea şabloanelor. SVM-ul primeşte mai multe instanţe ale datelor şi, pentru

fiecare în parte, specifică în ce categorie se încadrează. SVM este un clasificator binar liniar. Un SVM construieşte un hiperplan într-un spațiu *N*-dimensional care poate fi folosit pentru clasificări, regresii sau în alte scopuri. O separare bună se realizează de hiperplanul care are cea mai mare distanță față de cel mai apropiat punct din datele de test, din fiecare clasă. Acest punct poartă numele de margine funcțională; cu cât este mai mare marginea, cu atât eroarea de generalizare a clasificatorului este mai mică.

Problema detectării vandalismului pe Wikipedia folosind SVM a fost abordată de către Potthast et al. (2008). Datele de intrare se definesc ca un set de revizii, unde fiecare edit conține informații despre două revizii consecutive ale aceluiaşi articol. Definindu-se un set de funcții, fiecare edit este mapat într-un vector de *N* elemente, unde *N* este numărul de funcții, iar valoarea vectorului în acea dimensiune este rezultatul aplicării funcției asupra reviziei.

Învățarea bazată pe **arbori de decizie** foloseşte astfel de arbori ca un model predictiv care specifică valoarea unei date, bazându-se pe observații anterioare făcute asupra sa. Scopul este de a crea un model care poate estima valoarea unei variabile pe baza mai multor variabile de intrare. Fiecare nod interior al arborelui corespunde unei astfel de variabile şi de la fiecare nod se crează muchii în arbore pentru fiecare valoare posibilă a variabilei. Fiecare frunză specifică valoarea variabilei țintă ce se obține în funcție de valorile variabilelor de intrare prezente în drumul de la rădăcina arborelui până la frunza respectivă. Dacă la un moment dat valoarea din frunză nu este suficient de concludentă (adică suficient de apropiată de una dintre clasele posibile de apartenență a variabilei țintă), se continuă cu aplicarea altei variabile de intrare, adăugându-se un nou nivel de descendenți corespunzător nodului curent.

Adler et al. (2010) urmează o abordare bazată pe arbori de decizie în rezolvarea problemei de detecție a vandalismului pe Wikipedia. Variabila țintă este reprezentată de editarea analizată, iar variabilele de intrare sunt calculate pe baza conținuturilor versiunilor trecute ale articolului, profilul utilizatorului sau comentariile ataşate reviziei. În total se folosesc 15 noduri de decizie, printre care enumerăm: reputația utilizatorului, ora din zi în care a fost creată revizia, mărimea comentariului, histograma textului din versiunea trecută a articolului (histograma trecută), histograma curentă, etc. Clasificatorul ales este **ADTree** (bazat pe algoritmul propus de Freund şi Mason, 1999), al cărui API este integrat în setul de clasificatoare Weka. Acesta lucrează în modul următor: pe baza id-ului de revizie a articolului se

calculează valorile ce corespund celor 15 noduri ale arborelui, iar apartenența editului la una dintre cele două categorii este rezolvată în câteva milisecunde.

Harpalani et al. (2010) propun o soluție compusă de rezolvare a problemei de detecție a vandalismului pe Wikipedia, şi anume un hibrid între arbori de decizie şi clasificatori Bayesieni clasici, rezultând modelul **NBTree**. Pentru detectarea articolelor vandalizate sunt analizate 11 proprietăți directe sau indirecte (meta-informații) ale reviziilor şi se construieşte un arbore de decizie cu 8 noduri, pe baza exemplelor de antrenament. Ramurile arborelui rezultat determină drumul ales în clasificarea unei intrări, în funcție de deciziile luate în elaborarea drumului respectiv. Soluția aleasă diferă față de alte abordări bazate pe arbori de decizie prin faptul că atunci când se ajunge la o frunză, se aplică un clasificator clasic Naive Bayes, bazat pe un model specific frunzei respective. Deci, datele sunt dublu-clasificate: mai întâi arborele de decizie le partiționează în mod optim iar apoi sunt construite mai multe clasificatoare Bayesiene.

O *rețea Bayesiană* este un model probabilistic bazat pe grafuri orientate care reprezintă un set de valori aleatoare şi dependențele condiționate dintre ele sub forma unui graf orientat aciclic. De exemplu, o astfel de rețea poate reprezenta relațiile probabilistice dintre boli şi simptome; dându-se simptomele, rețeaua poate fi folosită pentru a calcula probabilitățile de existență a unor boli.

Din punct de vedere formal, rețelele Bayesiene sunt grafuri orientate aciclice ale căror noduri reprezintă valori aleatoare în sensul Bayesian: pot fi cantități observabile, variabile latente, parametri necunoscuți sau ipoteze. Muchiile reprezintă dependențe condiționate; nodurile care nu sunt conectate reprezintă variabile care sunt independente din punct de vedere al relației de condiționalitate. Fiecărui nod îi este asociată o funcție de probabilitate care are ca intrare un set de valori reprezentând valorile părintelui nodului şi are ca ieşire probabilitatea valorii reprezentate de nod. De exemplu, dacă părinții sunt m valori boolene, atunci funcția de probabilitate poate fi exprimată ca o tabelă cu 2^m intrări, câte una pentru fiecare din cele 2^m combinații posibile ale valorilor din nodurile părinte.

În domeniul statistic, **modelul logistic** (sau modelul bazat pe **regresii logistice**) este folosit pentru a prezice probabilitatea de desfăşurare a unui

eveniment analizând o funcție logistică (ce ia forma unei curbe logistice). Aceasta are următoarea definiție:

$$f(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

Unde:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ... + \beta_k x_k$$

Coeficienții β_i se numesc coeficienții regresiei şi fiecare dintre ei specifică dimensiunea contribuției lui x_i la factorul de decizie. Deci, pentru a folosi o astfel de regresie pentru problema acestei lucrări, trebuie definit un vector de variabile şi câte un coeficient de contribuție pentru fiecare.

Modelul logistic prezintă □i o extindere care implică anumite aspecte ale teoriei Bayesiene. Mai exact, considerând clasificatorul $y = f(x)$ si un set de date de antrenament $D = \{(x_1, y_1),...,(x_i, y_i),...,(x_n, y_n)\}$, se define□te vectorul $x_i = [x_{i,1},...,x_{i,j},...,x_{i,d}]^T$ ca fiind frecvențele cuvântului x_i în documentele *(1,d)*. Valorile lui y_i stabiliesc în ce masură cuvântul aparține de o clasă sau alta. Mai exact, se caută probabilitatea condiționată de forma:

$$p(y = +1 | \beta, x^i) = f(\beta^T x_i) = f(\sum_j \beta_j x_{i,j})$$

În expresia de mai sus, *f* este funcția logistică; astfel se face trecerea de la frecvențele cuvintelor (modelul Bayesian) la modelul regresiei logistice. $p(y = +1 | x_i)$ va reprezenta o estimare a probabilității ca al *i*-lea document să aparțină acestei categorii. O descriere detaliată a acestui tip de clasificare a fost propusă de către Genkin et al. (2004).

Toți algoritmii de clasificare prezentați în acest paragraf au ca scop încadrarea unei instanțe la o anumită clasă. Deoarece lucrează în moduri total diferite, rezultatele obținute vor fi interesant de comparat.

Evaluarea clasificatorilor va fi efectuată folosind framework-ul **Weka**. Acesta reprezintă o colecție de soluții software dedicată analizei datelor şi învățării automate pe baza acestei analize. Weka este disponibil gratuit sub licența GNU General Public License (licență publică generală) şi a fost dezvoltată la Universitatea din Waikato, Noua Zeelandă.

Colecțiile de cod sursă din Weka au fost scrise în Java, deci pot fi integrate în orice aplicație Java. Weka suportă mai multe operațiuni specifice de minerit al datelor, şi anume: preprocesare, clusterizare, clasificare, regresie, vizualizare, sau selectare a caracteristicilor.

Ne propunem folosirea Weka pentru a evalua toți clasificatorii prezentați anterior: **ADTree**, **NBTree**, **BayesNet**, **BayesLogisticRegression** şi **SVM**. Toți în afară de cel din urmă sunt implementați în interiorul librăriilor native Weka.

Pentru analiza soluției bazată pe SVM vom folosi Weka şi **LibSVM** (http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html) – un software integrat dezvoltat pentru clasificări bazate pe vectori suport. Acesta are ca scop facilitarea modului de implementare a soluțiilor bazate pe clasificatoare SVM. Printre altele, LibSVM rezolvă probleme de clasificări C-SVM, nu-SVM, one-class-SVM, sau regresii epsilon-SVM şi nu-SVM.

În aplicația WikiDetect este folosită o implementare LibSVM în Java cu interfața Weka numită **SVM Weka** (http://cns.bu.edu/~gsc/CN710/pmwiki.php?n=Main.SVMWeka).

4. Analiza rezultatelor

Pentru a atinge performanțele optime, au fost rulate o serie de teste în vederea obținerii unor rezultate cât mai bune pornind de la corpusul disponibil pentru evaluare. În cadrul acestui capitol descriem paşii efectuați în acest sens, împreună cu optimizările făcute i rezultatele obținute.

Pentru prima rulare completă a clasificatoarelor am calculat valorile caracteristicilor prezentate anterior pentru toate reviziile din corpusul de antrenament. În plus, am realizat o normalizare forțată a datelor, pe baza unor valori maxime empirice. De exemplu, am considerat că numărul maxim de caractere consecutive este 100, apoi am realizat normalizarea la intervalul [-1, 1] împărțind toate valorile obținute la 100. Aceeaşi procedură am aplicat-o în cazul mai multor caracteristici. Rezultale obținute după prima rulare sunt rezumate în *tabelul 1*.

BayesianLogisticRegression - Imposibil de aplicat

BayesNet (92.3987 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.951	0.418	0.967	0.951	0.959	0.921	0.0
0.582	0.049	0.476	0.582	0.524	0.921	1.0
Media ponderată						

0.924	0.392	0.932	0.924	0.927	0.921	

NBTree (94.4434 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.982	0.538	0.959	0.982	0.97	0.919	0.0
0.462	0.018	0.663	0.462	0.544	0.919	1.0
Media ponderată						
0.944	0.501	0.938	0.944	0.94	0.919	

SVM (93.1781 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.999	0.931	0.933	0.999	0.965	0.534	0.0
0.069	0.001	0.794	0.069	0.127	0.534	1.0
Media ponderată						
0.932	0.864	0.923	0.932	0.904	0.534	

ADTree (93.2881 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.978	0.652	0.951	0.978	0.964	0.915	0.0
0.348	0.022	0.553	0.348	0.427	0.915	1.0
Media ponderată						
0.933	0.607	0.922	0.933	0.926	0.915	

Tabelul 1. Rezultate obținute după prima rulare: normalizare empirică

Se observă faptul că clasificatorul BayesianLogisticRegresion nu a putut fi aplicat deoarece datele sunt incorect normalizate. Clasificatorii bazați pe arbori de decizie (în special NBTree) obțin rezultate destul de bune în ansamblu (94% date identificate corect), dar slabe din punctul de vedere al detectării editărilor vandalizate (0.66 precizie şi 0.46 acoperire pentru clasa 1.0 – revizii vandalizate). Clasificatorul SVM obține rezultate slabe prin prisma acoperirii clasei 1.0 şi, respectiv, a ariei de sub curba ROC.

Pentru a obține rezultate mai bune, am realizat o serie de modificări în calcularea valorilor vectorilor instanțelor. Astfel, pentru a doua rulare, am eliminat forma de normalizare de la prima rulare şi am introdus un filtru care elimină 75% dintre reviziile non-vandalizate. Acest lucru a fost necesar deoarece cele două clase (regular şi vandalism) erau nebalansate, existând un raport de aproximativ 9:1 între ele. Din acest motiv, clasificatoarele au rulat pe un număr de aproximativ 10.000 de instanțe. Rezultatele obținute după a doua rulare sunt prezentate în *tabelul 2*.

BayesianLogisticRegression - Imposibil de aplicat

BayesNet (85.785 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.889	0.241	0.92	0.889	0.905	0.913	0.0
0.759	0.111	0.686	0.759	0.721	0.913	1.0
Media ponderată						
0.858	0.21	0.864	0.858	0.86	0.913	

NBTree (85.7749 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.892	0.249	0.918	0.892	0.905	0.912	0.0
0.751	0.108	0.689	0.751	0.718	0.912	1.0
Media ponderată						
0.858	0.215	0.863	0.858	0.86	0.912	

SVM (34.4977 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.2	0.2	0.758	0.2	0.316	0.5	0.0
0.8	0.8	0.242	0.8	0.371	0.5	1.0
Media ponderată						
0.345	0.345	0.633	0.345	0.33	0.5	

ADTree (86.0575 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.901	0.265	0.914	0.901	0.907	0.908	0.0
0.735	0.099	0.702	0.735	0.718	0.908	1.0
Media ponderată						
0.861	0.225	0.863	0.861	0.862	0.908	

Tabelul 2. Rezultate obținute după a doua rulare: date nenormalizate, clase balansate

Din nou, din cauza nenormalizării datelor, clasificatorul BayesLogisticRegression nu a putut fi aplicat. BayesNet, ADTree şi NBTree obțin rezultate mult mai bune din punctul de vedere al identificării reviziilor vandalizate, mai exact precizia şi acoperirea depăşesc 0.7. SVM obține rezultate încă şi mai slabe decât cele de la prima rulare. Acest lucru se întâmplă deoarece vectorii de valori sunt practic incompatibili cu modul de lucru SVM care, pentru obținerea de rezultate corecte, necesită ca valorile atributelor să fie cât mai bine distribuite in domeniul [*valMin*,*valMax*].

Întrucât rezultatele nu au fost considerate satisfăcătoare nici în acest caz, pentru a treia rulare, au fost modificate valorile returnate de funcțiile care calculează valori ale atributelor care sunt centrate prea mult spre anumiți poli, cum ar fi anumite rapoarte care generează valori apropiate de 0. De asemenea, îndepărtăm toate intervalele de tipul [*x*, *valMax*] sau [*valMin*, *x*] pe care observăm preponderent valori ce denotă edituri curate, şi eliminăm filtrul de balansare a datelor (cu scopul îmbunănățirii rezultatelor corespunzătoare aplicării SVM). Rezultatele obținute după a treia rulare se observă în *tabelul 3*.

Se poate observa totu□i că a treia rulare necesită o durată prea mare de prelucrare şi nu prezintă rezultate mai bune decât rulările precedente. Din acest motive, s-a decis rebalansarea claselor pentru a patra rulare, ale cărei rezultate sunt rezumate în *tabelul 4*.

Rezultatele obținute după a patra rulare sunt bune (ADTree prezintă o arie ROC de 0.9, precizie şi acoperire ponderate de 0.85), însă aplicarea clasificatorului SVM necesită un timp prea mare iar BayesianLogisticRegression nu oferă rezultate satisfăcătoare. Astfel, pentru ultima rulare (rezultatele finale ale implementării sistemului) vom folosi filtrul de normalizare *weka.Filters.Unsupervised.Attribute.Normalize*.

BayesianLogisticRegression - Imposibil de aplicat

BayesNet (92.3364 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.955	0.477	0.962	0.955	0.958	0.9	0.0
0.523	0.045	0.482	0.523	0.502	0.9	1.0
Media ponderată						
0.923	0.445	0.926	0.923	0.925	0.9	

NBTree - Clasificatorul necesită un timp prea mare de prelucrare

SVM - Clasificatorul necesită un timp prea mare de prelucrare

ADTree (93.7513 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.992	0.746	0.943	0.992	0.967	0.897	0.0
0.254	0.008	0.716	0.254	0.375	0.897	1.0
Media ponderată						
0.938	0.691	0.927	0.938	0.923	0.897	

Tabelul 3. Rezultate obţinute după a treia rulare: date nenormalizate dar mai bine răspândite în domeniu, clase nebalansate

BayesianLogisticRegression (77.052 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.99	0.919	0.772	0.99	0.867	0.536	0.0
0.081	0.01	0.728	0.081	0.145	0.536	1.0
Media ponderată						
0.771	0.699	0.761	0.771	0.693	0.536	

BayesNet (84.5634 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.881	0.265	0.913	0.881	0.896	0.899	0.0

0.735	0.119	0.663	0.735	0.697	0.899	1.0
Media ponderată						
0.846	0.23	0.852	0.846	0.848	0.899	

NBTree (84.9571 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.896	0.295	0.905	0.896	0.9	0.895	0.0
0.705	0.104	0.683	0.705	0.694	0.895	1.0
Media ponderată						
0.85	0.249	0.851	0.85	0.85	0.895	

SVM - Clasificatorul necesită un timp prea mare de prelucrare

ADTree (85.0177 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.898	0.299	0.904	0.898	0.901	0.899	0.0
0.701	0.102	0.686	0.701	0.694	0.899	1.0
Media ponderată						
0.85	0.251	0.851	0.85	0.851	0.899	

Tabelul 4. Rezultate obtinute dupa a patra rulare: date nenormalizate dar mai bine raspandite in domeniu, clase balansate

După cum se observă, procedura de normalizare are un efect foarte mare asupra corectitudinii funcționării clasificatorului SVM. Deoarece acesta construieşte un hiperplan *N*-dimensional, trebuie să ne asigurăm că toate dimensiunile rămân la fel de „importante", ceea ce presupun normalizarea lor la un domeniu comun. De asemenea, normalizarea trebuie făcută independent de valorile celorlalte atribute, ceea ce înseamnă că ceea ce este considerat caracteristic unui act de vandalism într-un caz, poate avea o cu totul altă valoare într-un alt caz.

4.2 Rezultate finale

Am considerat că rezultatele finale ale aplicației (expuse în *tabelul 5*) trebuie să fie punctul de întâlnire al tuturor celor cinci algoritmi de

clasificare. Luați separat, însă, unii din ei obțin rezultate mai bune după alte scenarii de rulare, descrise în paragraful anterior.

BayesianLogisticRegression (79.5457 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.839	0.341	0.885	0.839	0.862	0.749	0.0
0.659	0.161	0.566	0.659	0.609	0.749	1.0
Media ponderată						
0.795	0.298	0.808	0.795	0.8	0.749	

BayesNet (84.5634 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.881	0.265	0.913	0.881	0.896	0.899	0.0
0.735	0.119	0.663	0.735	0.697	0.899	1.0
Media ponderată						
0.846	0.23	0.852	0.846	0.848	0.899	

NBTree (85.0076 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.897	0.297	0.905	0.897	0.901	0.896	0.0
0.703	0.103	0.685	0.703	0.694	0.896	1.0
Media ponderată						
0.85	0.25	0.851	0.85	0.851	0.896	

SVM (79.6164 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.836	0.33	0.888	0.836	0.862	0.753	0.0
0.67	0.164	0.566	0.67	0.614	0.753	1.0
Media ponderată						
0.796	0.289	0.811	0.796	0.802	0.753	

ADTree (85.0177 % identificate corect)

Rata TP	Rata FP	Precizie	Acoperire	Metrica F	Aria ROC	Clasa
0.898	0.299	0.904	0.898	0.901	0.899	0.0
0.701	0.102	0.686	0.701	0.694	0.899	1.0
Media ponderată						
0.85	0.251	0.851	0.85	0.851	0.899	

Tabelul 5. Rezultate finale: date normalizate bine răspândite în domeniu, clase balansate

În *figura 2* se poate observa structura arborelui de decizie obținut în urma aplicării clasificatorului ADTree. După cum era de aşteptat, cele mai definitorii trăsături ale unui edit vandalizat sunt: Este_anonim, Numărul_cuvintelor_suspecte, Max_cuvânt şi Creşterea_dimensiunii. De asemenea, în *figura 3* este ilustrată curba ROC ce rezultă în urma aplicării aceluiaşi clasificator; aria de sub curbă reprezintă o metrică foarte importantă care contribuie la atribuirea unui grad înalt de încredere algoritmului de separație.

Considerăm că rezultatele obținute în urma aplicării clasificatorilor descrişi în capitolul precedent sunt comparabile cu cele obținute în alte lucrări care şi-au propus implementarea unui sistem de detecție automată a vandalismului pe Wikipedia. De exemplu, în cazul aplicării ADTree se observă coeficientul de 0.85 atât pentru precizie, cât şi pentru acoperire, valori superioare celor obținute de Adler et al. (2010), şi anume de 0.48, respectiv, 0.83. De asemenea, WikiDetect prezintă o precizie medie de 0.81 şi acoperire medie de 0.79 în urma aplicării SVM, coeficienți mai buni decât cei obținuți de West et al. (2010): 0.49, respectiv 0.5. Am centralizat mai multe rezultate din astfel de lucrări în tabelul 6. Menționăm faptul că rezultatele nu au fost obținute pe acelaşi set de date şi, deci, au doar un caracter informativ, şi nu compararativ.

Metoda	**Precizie**	**Acoperire**	**Observații**
SVM	49%	50%	1,3 milioane revizii dintre care 128 mii erau vandalizate (West et al., 2010)
Naive Bayes	58%	36%	370 mii revizii. Valorile din

	(41%)	(56%)	parenteză au fost obtinute luând în considerare □i probabilităţile iniţiale (Smets et al., 2008)
Regresie logistică	83%	55%	500 mii revizii (Belani, 2009)
Compresia Markov dinamică	86%	56%	5768 revizii. Rezultate obţinute pentru detectarea vandalizării prin inserţie (Itakura şi Clarke, 2009)
Vizualizări persistente ale cuvintelor	77%	62%	676 revizii. Algoritmul nu are ca ţintă precizia/acoperirea, ci rapiditatea cu care se repară un incident de vandalism (Priedhorsky et al., 2007)
ADTree	48%	83%	317 mii revizii dintre care 15 mii erau vandalizate (Adler et al., 2010)
NBTree	61%	25%	15 mii revizii dintre care 944 erau vandalizate (Harpalani et al., 2010)

Tabelul 6. Centralizarea rezultatelor obţinute în alte lucrări de specialitate

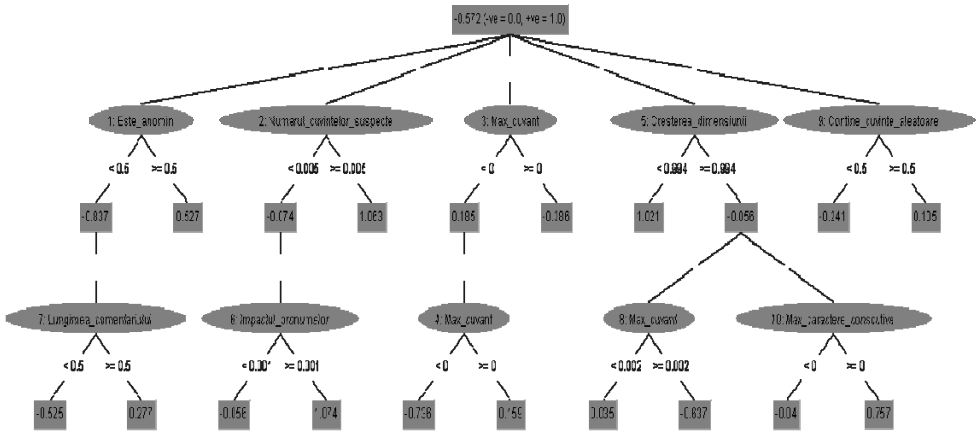

Figura 2. Structura arborelui de decizie obţinut în urma aplicării ADTree

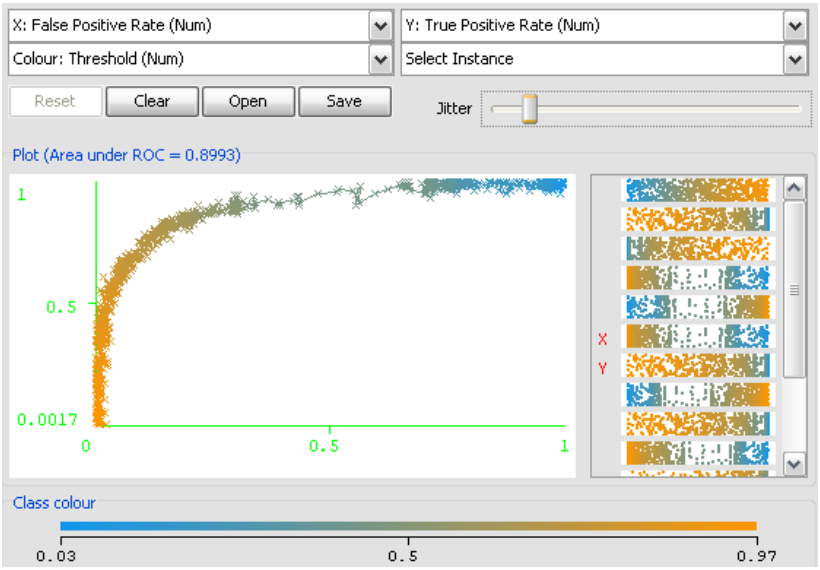

Figura 3. Curba ROC rezultată în urma aplicării ADTree

Pe baza rezultatelor obținute am calculat aria de sub curba precizie – acoperire (PR-AUC) pentru fiecare dintre cele două clase, folosind aplicația Java *AUCCalculator* (http://mark.goadrich.com/programs/AUC/). S-au generat valorile 0.96 pentru clasa 0 şi 0.74 pentru clasa 1. În tabelul 7 observăm că WikiDetect a produs rezultate comparabile cu cele mai bune trei aplicații de la conferința PAN-10 (prezentate de Potthast, Stein şi Holfeld, 2010).

ROC-AUC	**PR-AUC**	**Detector**
0.92236	0.66522	Mola Velasco, 2010
0.90351	0.49263	Adler et al., 2010
0.89930	*0.74442*	*WikiDetect*
0.89856	0.44756	Javanmardi, 2010

Tabelul 7. Rezultatele WikiDetect în comparație cu cele mai bune detectoare de la PAN-10

5. Concluzii şi direcţii viitoare

Lucrarea de faţă a prezentat detaliile de proiectare şi implementare a unei soluţii de detecţie automată a vandalismului pe Wikipedia. Au fost folosiţi o serie de algoritmi de separare ale căror rezultate au fost comparate şi analizate.

În urma testelor, putem concluziona că soluţia ajunge la rezultate convingătoare, cele mai bune fiind obţinute în urma aplicării clasificatorului ADTree (aria de sub curba ROC ~0.9). Deoarece ne-am concentrat mai mult pe obţinerea unor rezultate bune folosind mai multe clasificatoare, putem afirma că eficienţa aplicaţiei ar fi putut creşte considerabil dacă am fi ales un singur clasificator (de exemplu, ADTree sau NBTree) şi am fi construit implementarea pe baza acestuia.

Scopul activităţii de cercetare a fost de a prelucra un corpus oficial de date şi de a compara rezultatele obţinute prin aplicarea mai multor clasificatoare. Acest lucru a fost îndeplinit prin modificarea setărilor de lucru şi s-a ajuns în final la o soluţie uniform pozitivă şi aplicabilă într-un mediu real.

Soluţia ia în considerare şi motivaţia utilizatorilor rău-intenţionaţi, detecţia automată a editărilor, precum şi categoriile de vandalism descrise în mod oficial de către Wikipedia. Faptul că acestea nu au fost construite pentru integrarea într-un sistem software de detecţie automată face ca trecerea de la marcarea manuală la cea automată să prespună implementarea de tehnici complexe de învăţare automată, ceea ce considerăm că am realizat cu succes.

Ca o direcţie viitoare, menţionăm posiblitatea de a combina doi sau mai mulţi clasificatori în vederea obţinerii unei acoperiri mai mari a reviziilor vandalizate. În mod normal, acest fapt conduce la prelucrarea într-un mod diferit a valorilor normalizate ce vor compune vectorii instanţă.

O altă îmbunătăţire ulterioară constă în maximizarea ariei de sub curba ROC prin metode suplimentare cum ar fi adăugarea de noi caracteristici, eliminarea valorilor aberante sau singulare, sau includerea în analiză a meta-datelor ce descriu adnotatorii datelor de antrenament.

În final, trebuie avut în vedere că problema discutată are valoare practică imediată deoarece este o problemă reală cu o complexitate ridicată şi efecte în timp continuu cu care se confruntă moderatorii voluntari ai Wikipedia. Din acest motiv, orice îmbunătăţire ulterioară va avea efecte pozitive în

cadrul procedurii de separație, ținând cont de faptul că rata de editare pe Wikipedia este în continuă creştere.

Referințe

Adler B.T., de Alfaro L., Pye I. *Detecting Wikipedia Vandalism using WikiTrust*. Proceedings of the 2010 Conference on Multilingual and Multimodal Information Access Evaluation, 2010.

Belani A. *Vandalism Detection in Wikipedia: a Bag-of-Words Classifier Approach*. The Computing Research Repository, vol. 1001, 2009.

Ciucanu, A., Dănilă, A., Nechifor, P., Iftene, A. *Detectarea vandalismului pe Wikipedia*. Proceedings of RoCHI 2010, 2010.

Freund Y., Mason L. *The Alternating Decision Tree Algorithm.* Proceedings of the 16th International Conference on Machine Learning, 1999.

Genkin A., Lewis D., Madigan D. *Large-Scale Bayesian Logistic Regression for Text Categorization*. Technometrics 49, 2004.

Harpalani M., Phumprao T., Bassi M., Hart M., Johnson R. *Wiki Vandalysis - Wikipedia Vandalism Analysis*. Proceedings of the 2010 Conference on Multilingual and Multimodal Information Access Evaluation, 2010.

Itakura K.Y., Clarke C. *Using Dynamic Markov Compression to Detect Vandalism in the Wikipedia*. Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, 2009.

Javanmardi S. *Vandalism detection in Wikipedia: a high-performing, feature-rich model and its reduction through Lasso*. Proceedings of the 7th International Symposium on Wikis and Open Collaboration, 2011

Mola Velasco S.M. *Wikipedia Vandalism Detection Through Machine Learning: Feature Review and New Proposals*. Notebook Papers of CLEF 2010 LABs and Workshops, 2010

Potthast M. *Crowdsourcing a Wikipedia Vandalism Corpus*. Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval, 2010.

Potthast M., Stein B., Gerling R. *Automatic Vandalism Detection in Wikipedia*. Lecture Notes in Computer Science Volume 4956, 2008.

Potthast M., Stein B., Holfeld T. *Overview of the 1st International Competition on Wikipedia Vandalism Detection*. Notebook Papers of CLEF 2010 LABs and Workshops, 2010.

Priedhorsky R., Chen J., Lam S.K., Panciera K., Terveen L., Riedl J. *Creating, Destroying, and Restoring Value in Wikipedia*. Proceedings of the 2007 international ACM conference on Supporting group work, 2007.

Smets K., Goethals B., Verdonk B. *Automatic Vandalism Detection in Wikipedia: Towards a Machine Learning Approach*. Proceedings of the Workshop on Wikipedia and

Artificial Intelligence: An Evolving Synergy, 2008.

Vapnik V., Cortes C. *Support-Vector Networks*. Machine Learning, vol. 20, nr. 3, 1995.

West, A.G., Kannan S., Lee I. *Detecting Wikipedia Vandalism via Spatio-Temporal Analysis of Revision Metadata*. Proceedings of the Third European Workshop on System Security EUROSEC, 2010.