



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE**

Extragerea de informație structurată din descrieri în limbaj natural ale endoscoپیilor folosind algoritmul Apriori

LUCRARE DE LICENȚĂ

Absolvent: **Alexandra Ioana BALI**

Coordonator științific: **S.L. dr. ing. Radu Răzvan SLĂVESCU**

2015



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE**

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Alexandra Ioana BALI**

**Extragere de informație structurată din descrieri în limbaj natural ale endoscopilor
folosind algoritmul Apriori**

1. **Enunțul temei:** *Să se dezvolte un sistem software capabil să extragă în mod automat informație structurată din texte în limbaj natural, în română, care descriu endoscopii. Sistemul va adnota aceste descrieri folosind etichete dintr-un set stabilit. Regulile de adnotare sunt învățate de către sistem, pornind de la un set de texte în limbaj natural deja etichetate. Învățarea regulilor se va face semi-automat, în sensul că sistemul propune un set de reguli posibil de aplicat, iar utilizatorul le selectează pe cele pe care le consideră corecte.*
2. **Conținutul lucrării:** *Lucrarea este alcătuită din 8 capitole, bibliografie și anexă. Capitolele sunt: Introducere, Obiectivele și Specificațiile Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii.*
3. **Locul documentării:** Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanți:**
5. **Data emiterii temei:** 1 noiembrie 2014
6. **Data predării:** 19 Iunie 2015

Absolvent:

Alexandra Ioana BALI

Coordonator științific:

S.L. dr. ing. Radu Răzvan
SLĂVESCU

**UNIVERSITATEA TEHNICĂ**

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE****Declarație pe proprie răspundere privind
autenticitatea lucrării de licență**

Subsemnatul(a) **Bali Alexandra Ioana**, legitimat(ă) cu **CI** seria **SM** nr. **495893** CNP **2920101303310**, autorul lucrării **Extragere de informație structurată din descrieri în limbaj natural ale endoscopiilor folosind algoritmul Apriori** în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea **Tehnologia Informației** din cadrul Universității Tehnice din Cluj-Napoca, sesiunea **iulie 2014** a anului universitar **2014-2015**, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data

19.06.2015

Nume, Prenume

BALI Alexandra Ioana

Semnătura

Instrucțiuni generale

Cuprins

Capitolul 1. Introducere	1
1.1. Contextul proiectului	1
1.2. Structura lucrării	1
Capitolul 2. Obiectivele și Specificațiile Proiectului.....	3
2.1. Enunțul temei.....	3
2.2. Obiectivele proiectului.....	3
2.3. Cerințele proiectului	4
2.3.1. Cerințe funcționale	5
2.3.2. Cerințe non-funcționale	6
Capitolul 3. Studiu Bibliografic.....	7
3.1. Procesarea limbajului natural	7
3.2. Data mining	8
3.2.1. Prezentare generală.....	8
3.2.2. Frequent Pattern Mining	9
3.2.3. Algoritmul Apriori.....	10
3.2.4. Alți algoritmi	11
3.3. Procesarea textelor.....	12
3.3.1. Procesare texte cu parsare lingvistică	12
3.3.2. Procesare texte fără parsare lingvistică	14
3.4. Alte studii și realizări.....	15
Capitolul 4. Analiză și Fundamentare Teoretică.....	17
4.1. Arhitectura conceptuală a sistemului.....	17
4.1.1. Pattern-ul arhitectural Model-View-Controller	17
4.1.2. Prezentarea arhitecturii conceptuale	18
4.1.3. Modulul de procesare al abrevierilor	18
4.1.4. Modulul de generare de reguli	20
4.1.5. Modulul de adnotare	22
4.2. Modelul de date	23
4.2.1. Prezentare generală.....	23
4.2.2. Datele procesul de învățare.....	24
4.2.3. Datele procesul de adnotare.....	25
4.3. Algoritmul Apriori.....	25

Tabel 4.2 Exemplu de tranzacții	25
Capitolul 5. Proiectare de Detaliu si Implementare	29
5.1. Implementarea arhitecturii sistemului	29
5.1.1. Vederea generală	29
5.1.2. Implementarea pattern-ului Model-View-Controller	30
5.1.3. Implementarea modului de abrevieri	31
5.1.4. Implementarea modului de generare de reguli	33
5.1.5. Implementarea modului de adnotare	35
5.2. Modulul de prelucrare a datelor.....	37
5.2.1. Prelucrarea datelor de învățare	38
5.2.2. Algoritmul de înlocuire al abrevierilor	39
5.2.3. Procesul de adnotare al datelor	41
5.2.4. Modul de integrare al algoritmului Apriori	42
Capitolul 6. Testare și Validare	44
6.1. Testarea generală a codului	44
6.2. Testare funcțională.....	44
6.2.1. Verificarea funcționalității.....	44
6.2.2. Validarea datelor generate	45
6.3. Testare funcțională.....	45
6.3.1. Impactul setului de date de învățare	45
6.3.2. Impactul mai multor iterații ale algoritmului Apriori.....	46
6.3.3. Variații ale suportului	47
6.3.4. Testarea înlocuirii abrevierilor	48
Capitolul 7. Manual de Instalare si Utilizare	50
7.1. Manual de instalare.....	50
7.2. Manual de utilizare	50
7.2.1. Interfața modului de adăugare și actualizare a abrevierilor	51
7.2.2. Interfața modului de învățare și generare de reguli	52
7.2.3. Interfața modului de adnotare.....	53
Capitolul 8. Concluzii	55
8.1.1. Rezultate obținute	55
8.1.2. Dezvoltări ulterioare	56
Bibliografie	58
Anexa (1)	59

Capitolul 1. Introducere

1.1. Contextul proiectului

În toate perioadele istorice și de-a lungul timpului medicina a beneficiat cu prioritate de cele mai noi și performante achiziții ale științei și tehnologiei. Chiar din cele mai vechi timpuri s-au identificat numeroase specii de plante cu posibilități curative, s-au folosit cele mai noi materiale disponibile pentru instrumente medicale și s-a încercat mereu ca medicina să fie la zi cu cele mai noi tehnologii din acest domeniu. În prezent sistemele informatice au un mare impact în medicină, ajungându-se până în punctul în care s-a delimitat un nou domeniu aparte, numit informatică medicală, care este dedicat studiului utilizării informaticii în medicină. În acest fel, se pot aplica diferite tehnici de calcul în aproape orice domeniu medical, obținându-se o varietate de clase de aplicații care au utilizări în sistemul medical. Printre primele astfel de aplicații au fost acelea care au încercat punerea de diagnostic în urma unor rezultate obținute după anumite examinări, întrucât diagnosticarea creează unele dintre cele mai serioase probleme. În acest fel, dezvoltarea de programe de diagnosticare a devenit un domeniu de cercetare bine definit în informatica medicală, la scurt timp după conturarea inteligenței artificiale ca și ramură aparte a informaticii.

Unele dintre cele mai importante influențe din medicină le reprezintă limbajul natural, întrucât este de departe cel mai convenabil mod în care personalul medical poate transmite și reține informații. Ca o consecință a acestui lucru, o mare parte dintre informațiile medicale importante nu sunt disponibile într-o formă sau structură anume. De exemplu, înregistrările despre pacienți din majoritatea instituțiilor constau în volume mari de text nestructurat. Acest lucru indică o nevoie tot mai mare de utilizare a procesării automate a limbajului natural în acest domeniu. Astfel de sisteme de procesare au posibilitatea de a captura date medicale scrise în limbaj natural și de a le structura în diferite moduri. După structurare, acestea ar putea fi salvate ca înregistrări permanente și regăsite cu ajutorul unor interogări, după care ar putea fi afișate utilizatorilor chiar și în limbaj natural. Chiar dacă încă nu există un asemenea sistem care să realizeze toate aceste funcționalități într-o manieră integrată, multe dintre aceste funcții pot fi realizate separat cu ajutorul tehnologiei din prezent. Printre tehnicile utilizate în acest scop se numără folosirea metodelor statistice și a celor de pattern-matching.

1.2. Structura lucrării

Lucrarea elaborată este împărțită în 8 capitole, fiecare dintre acestea cuprinzând cele mai relevante aspecte cu privire la aplicația implementată. La sfârșitul lucrării sunt prezente de asemenea și secțiunile Bibliografie și Anexa. În continuare voi descrie pe scurt ce conține fiecare capitol.

Capitolul 1. Introducere - se enunță tema acestei lucrări și se descrie tema în contextul mai general al domeniului. Se descrie structura proiectului.

Capitolul 2. Obiectivele și Specificațiile Proiectului – în acest capitol este descris ce își propune proiectul, care sunt obiectivele care se doresc a fi atinse, cerințele funcționale și non-funcționale.

Capitolul 3. Studiu Bibliografic – sunt prezentate pe scurt tehnologiile și noțiunile întâlnite pe parcursul etapei de documentare, precum și alte abordări și soluții din alte domenii.

Capitolul 4. Analiză și Fundamentare Teoretică – acest capitol se axează pe descrierea conceptuală a arhitecturii, componentelor, algoritmilor și modelului de date. Tot în acest capitol sunt detaliate conceptele teoretice folosite în cadrul implementării soluției propuse.

Capitolul 5. Proiectare de Detaliu și Implementare – este dedicat sublinierii etapei de proiectare și implementare a sistemului, urmată de o detaliere atentă a tuturor componentelor și justificarea soluției alese.

Capitolul 6. Testare și Validare – prezentarea rezultatelor obținute efectuând diferite teste.

Capitolul 7. Manual de utilizare – prezentarea resurselor de care are sistemul nevoie pentru a funcționa, care sunt componentele necesare pentru instalare, cum se instalează și cum funcționează sistemul.

Capitolul 8. Concluzii – descrie analiza rezultatelor obținute și atingerea obiectivelor propuse în Capitolul 2. Tot aici sunt prezentate posibilele dezvoltări ulterioare și îmbunătățiri ale acestei lucrări.

Bibliografie – cuprinde resursele utilizate în realizarea studiului bibliografic și în proiectarea și dezvoltarea acestei aplicații.

Anexa – conține o publicație realizată pe baza acestei lucrări și a aplicației proiectate.

Capitolul 2. Obiectivele și Specificațiile Proiectului

2.1. Enunțul temei

Să se dezvolte un sistem software capabil să extragă în mod automat informație structurată din texte în limbaj natural, în română, care descriu endoscopii. Sistemul va adnota aceste descrieri folosind etichete dintr-un set stabilit. Regulile de adnotare sunt învățate de către sistem, pornind de la un set de texte în limbaj natural deja etichetate. Învățarea regulilor se va face semi-automat, în sensul că sistemul propune un set de reguli posibil de aplicat, iar utilizatorul le selectează pe cele pe care le consideră corecte.

2.2. Obiectivele proiectului

Principalul obiectiv pe care îl urmărește acest proiect este realizarea unei aplicații de procesare a limbajului natural care să fie capabilă să extragă informații din texte scrise în limba română, rezultate în urma examinărilor endoscopice. Acest proces trebuie să aibă loc în mod automat, în sensul că nu este nevoie de intervenția explicită a utilizatorului uman pe parcursul procesului. Extragerea informației are loc pe baza unor reguli învățate în prealabil de către sistem, nefiind nevoie astfel de parsarea lingvistică a observațiilor medicale. Aceste reguli se vor aplica pe setul de date de adnotat, iar la potrivirea uneia dintre ele cu o anumită observație, se va face adnotarea acelei înregistrări cu eticheta corespunzătoare.

Un exemplu de astfel de observații medicale deja adnotate este prezentat mai jos:

Tabel 2.1 – exemplu de observatii medicale adnotate

Stomac	S-Lum	S-Cont	S-MucANT	S-MucJAC	S-MucCO RP	SlezANT	S-LezJAC	S-LezCO RP	Pilor
Stomac cu antru nodular	N	N	NO	N	N	N	N	N	N
Nodular gastric antrum	N	N	NO	N	N	N	N	N	N

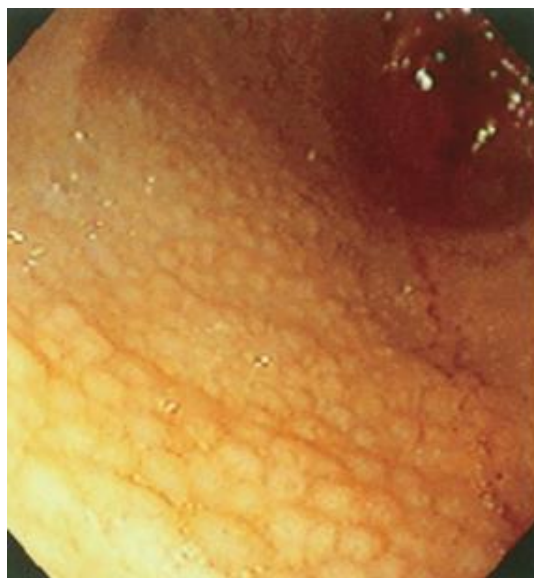


Figura 2.1 – stomacul din observatiile de mai sus

Așadar, un alt obiectiv avut în vedere în realizarea acestui proiect a fost de a-l proiecta astfel încât să fie capabil să învețe reguli de adnotare cât mai corecte și mai relevante, pe baza unui set de date în limbaj natural deja etichetate. Spre deosebire de procesul anterior, învățarea de reguli nu se face în mod automat ci semi-automat, adică este nevoie și de intervenția utilizatorului în timpul generării acestor reguli. Totuși, se dorește ca sistemul să știe să aleagă cele mai relevante reguli, făcând astfel muncă utilizatorului mai ușoară. Acest lucru se va realiza prin impunerea unei limite minime a support-ului de care se va ține cont la generare, în sensul că doar acele reguli care au support-ul mai mare decât limita vor fi luate în considerare.

Pe lângă utilizarea regulilor în scopul adnotării observațiilor, se mai dorește și înlocuirea cuvintelor prescurtate din cadrul acestora, așadar al treilea obiectiv pe care această aplicație își propune să îl îndeplinească este realizarea unui algoritm cât mai bun și mai eficient de înlocuire a abrevierilor care apar în cadrul observațiilor medicale. Pentru a ști cu ce trebuie să se înlocuiască fiecare abreviere, se va construi un dicționar cu înregistrări de tipul : abreviere – semnificație, pe baza căruia se vor face înlocuirile.

Nu în ultimul rând, acest sistem mai are ca obiectiv prezentarea functionalitatilor într-o manieră cât mai prietenoasă și mai ușor de folosit de către utilizatori, întrucât aceștia nu vor fi experți în domeniul IT. De asemenea, se dorește și ca sistemul să fie cât mai performant, să aibă timp bun de răspuns și să nu conțină erori.

2.3. Cerințele proiectului

În cadrul oricărui proiect, atât cerințele funcționale cât și cele non-funcționale au un rol important și anume, de a ajuta în proiectarea unui program funcțional corect, care să se comporte conform specificațiilor. În continuare voi descrie principalele cerințe funcționale și non-funcționale din cadrul proiectului.

2.3.1. Cerințe funcționale

Cerințele funcționale descriu funcționalitatea sistemului și afirmații despre serviciile pe care acesta trebuie să le conțină, cum trebuie el să răspundă la anumite intrări și cum să reacționeze în anumite situații. Principalele cerințe funcționale pe care acest proiect a trebuit să le respecte cu succes sunt:

- abilitatea sistemului de înlocuire a abrevierilor, cuvintelor prescurtate sau notațiile ambigue din cadrul observațiilor rezultate în urma examinării endoscopiei
- crearea unui modul care să se ocupe cu manipularea unui mic dicționar care să cuprindă pentru fiecare abreviere identificată semnificația acesteia
- abilitatea sistemului de a procesa un set de date în scopul învățării
- posibilitatea de a genera și manipula noi reguli cu ajutorul algoritmului Apriori
- adnotarea textelor medicale fără a utiliza o parsare lingvistică

Abilitatea sistemului de înlocuire a abrevierilor

Această cerință funcțională ocupă un loc important în cadrul acestui proiect, deoarece toate operațiile care urmează a fi efectuate se bazează pe rezultatele obținute la acest pas. Având în vedere rezultatele obținute în urma examinării endoscopiei unui pacient, acestea cuprind pe lângă multitudinea de cuvinte cheie sau termeni specifici, o serie de cuvinte prescurtate, notații ambigue și o mulțime de alte abrevieri. Înainte ca aceste informații să fie prelucrate, să treacă prin celelalte etape ale aplicației, mai întâi trebuie să treacă printr-un proces de înlocuire a abrevierilor.

Procesul de înlocuire a abrevierilor se bazează pe o altă cerință funcțională și anume *crearea unui mic dicționar în care se vor păstra abrevierile*. Astfel, această soluție aleasă va fi structurată în stilul clasic, conținând înregistrări de tip abreviere – semnificație. Pentru fiecare prescurtare sau notație care poate fi întâlnită în setul de date cu care lucrăm, se vor păstra înregistrări care să conțină prescurtarea respectivă împreună cu semnificația acesteia. Pentru o manipulare cât mai bună a acestui dicționar, am ales construirea unui modul asupra căruia să se efectueze și alte operații, printre care se enumeră : adăugarea unei noi înregistrări (abreviere - semnificație) , ștergerea unei înregistrări existente, salvarea modificărilor efectuate, căutarea după o anumită abreviere.

Posibilitatea de a genera și manipula noi reguli

Un modul important din cadrul acestui proiect constă în etapa de învățare. Această etapă presupune generarea unor mulțimi de reguli bazate pe un set de date bine definit. Acest modul pune la dispoziția utilizatorului posibilitatea de a alege setul datelor de intrare prin selectarea anumitor cuvinte cheie. O dată ce setul datelor de intrare este stabilit, se pot genera noi reguli pentru a putea fi validate. Acest proces de generare se va realiza cu ajutorul algoritmului Apriori, care pe baza unor parametrii va genera un număr X de reguli.

Adnotarea textelor medicale fără a utiliza o parsare lingvistică

Utumul pas care trebuie îndeplinit și unul dintre cei mai importanți, este procesul de adnotare a textelor medicale. Asemenea ca și la procesul de învățare, această etapă

necesita un set de date de intrare, asupra cărora să se efectueze operațiile din cadrul acestui modul. Primul pas care trebuie efectuat în acest proces constă în prelucrarea observațiilor medicale prin înlocuirea abrevierilor bazat pe “Dicționarul de abrevieri”. Al doilea pas este adnotarea observațiilor obținute la pasul anterior cu ajutorul regulilor obținute în procesul de învățare.

2.3.2. Cerințe non-funcționale

Cerințele non-funcționale sunt acele cerințe care nu sunt legate de funcții ale sistemului, sunt constrângeri ale serviciilor și funcțiilor oferite de un sistem. Amintim câteva cerințe non-funcționale pe care le îndeplinește și acest proiect.

Usability (utilizare ușoară) – aplicația va prezenta o interfață grafică prietenoasă și ușor de utilizat. Fiecare modul va înfățișa o interfață care cuprinde butoane ale căror nume sunt foarte sugestive.

Reliability (fiabilitate) – dacă vor fi atinse cu succes toate cerințele funcționale ale proiectului, acesta urmează să funcționeze corect și conform așteptărilor. În urma testelor efectuate asupra aplicației se va constata dacă aceasta a funcționat fără erori pentru o perioadă îndelungată de timp, ceea ce ne va confirma fiabilitatea acesteia.

Scalability (scalabilitate) – pe parcursul implementării acestui sistem vom avea grijă la alegerea structurilor de date folosite astfel încât acestea să fie eficiente din punctul de vedere al consumului de memorie. De asemenea, algoritmii folosiți nu sunt de o complexitate mare și ar trebui să poată procesa cu succes volume mari de date.

Maintainability (mentenanța) – această cerință va fi asigurată prin proiectarea și implementarea aplicației respectând un pattern arhitectural – Model-View-Controller. Astfel, sistemul va fi unul ușor de întreținut și extins, dacă va fi cazul.

Capitolul 3. Studiu Bibliografic

3.1. Procesarea limbajului natural

Procesarea limbajului natural (eng. Natural Language Processing), sau lingvistica computațională, este o știință interdisciplinară care combină informatica, inteligența artificială și lingvistica. Prelucrarea limbajului natural este o componentă a Inteligenței Artificiale (eng. Artificial Intelligence) și are ca principal scop cercetarea limbajelor scrise și vorbite.

În 1950, Alan Turing a publicat celebrul său articol “Computing Machinery and Intelligence” în care a descris cunoscutul test Turing: Într-o cameră se află un robot și un om, fiecare putând comunica cu exteriorul cu ajutorul unei tastaturi. Un observator neutru, aflat într-o cameră alăturată, conversează cu cei doi, tot prin intermediul unei tastaturi. Robotul este la fel de inteligent ca un om dacă în timpul conversației observatorul neutru nu poate face diferența dintre om și robot.

În prezent, numeroase instituții academice și mari corporații studiază prelucrarea limbajului natural, pregătind calculatoarele să treacă testul Turing. Scopul final este ca oamenii să poată vorbi cu calculatorul folosind limbajul liber, fără a cunoaște un limbaj de programare. Totodată și calculatorul trebuie să poată să își comunice rezultatele în limbaj natural.

Dezvoltarea de aplicații bazate pe prelucrarea limbajului natural este o provocare, deoarece calculatoarele necesită în mod normal ca oamenii să comunice cu acestea într-un limbaj de programare, întrucât acest tip de limbaj este precis, foarte structurat și nu este ambiguu. O alternativă de comunicare ar fi enunțarea unui număr limitat de comenzi vocale pe care calculatorul să le interpreteze, însă nimic complicat. Cu toate acestea, vorbirea umană nu este întotdeauna precisă. Deseori, limbajul natural are o doză de ambiguitate iar structura lingvistică poate depinde de mai multe variabile complexe, printre care se numără dialectele regionale, contextul social sau accentul.

Abordările actuale ale limbajului natural de procesare se bazează în general pe “machine learning”, o ramură a inteligenței artificiale care se ocupă cu studiul algoritmilor capabili să învețe de pe urma datelor și să facă preziceri asupra acestora. Paradigma “machine learning” este oarecum diferită față de încercările anterioare de procesare a limbajului. Celelalte implementări implică în general crearea manuală de seturi mari de reguli, care se vor aplica pe datele de procesat. În cazul machine-learning, în schimb, se utilizează algoritmi capabili să învețe în mod automat asemenea reguli în urma analizării unor seturi mari de exemple din lumea reală. Aceste exemple trebuie să fie în prealabil adnotate, astfel încât algoritmii să poată învăța reguli de pe urma lor.

În cadrul procesării limbajului natural s-au aplicat o serie numeroasă de clase de algoritmi “machine learning”. Printre primii utilizați s-au numărat și “arborii de decizie”, care produceau sisteme bazate pe reguli “if-then”, în mod similar cu regulile scrise manual. Odată cu trecerea timpului, cercetările s-au axat pe modele statistice care să genereze decizii probabilistice bazate pe atașarea anumitor costuri asupra datelor de intrare. Asemenea modele au avantajul că pot exprima certitudinea relativă a numeroase răspunsuri posibile diferite, în loc de a genera unul singur, ceea ce face ca algoritmii să producă rezultate mai fiabile.

Capacitatea unui calculator de a înțelege limbajul natural este foarte relevantă în domeniul medical. Există numeroase studii medicale, studii de chimie, biochimie, biologie ale căror rezultate ar putea fi folosite în diferite situații:

- companiile farmaceutice pot folosi rezultatele în procesul creării unor noi medicamente
- doctorii pot folosi rezultatele în prescrierea medicamentelor deja existente pe piață, în stabilirea diagnosticelor sau a metodelor de tratament.

Un calculator care înțelege limbajul natural poate compara eficient și rațional rezultatele studiilor de cercetare, poate stabili felul în care studiile de cercetare se completează reciproc, poate stabili dacă un studiu de cercetare confirmă sau contrazice rezultatele unor studii anterioare.

Un robot care trece testul Turing poate, în teorie, lua decizii cel puțin la fel de inteligente precum deciziile unui doctor care are acces la toate bazele de cunoștințe în domeniul medical.

3.2. Data mining

3.2.1. Prezentare generală

Data mining este un ansamblu de algoritmi și metode destinate explorării și analizei unor baze de date mari pentru identificarea din date de reguli, de asociații, de tendințe necunoscute, de structuri particulare ce caracterizează în manieră concisă esențialul informației utile. Data mining este analiza seturilor de date (deseori de dimensiuni mari) rezultate prin observații pentru a găsi relații noi și pentru sumarizarea datelor în moduri care sunt atât ușor de înțeles cât și utile celui ce deține datele. Un mecanism de data mining este prezentat în lucrarea publicată de Scoot Spangler și Jeffrey Kreulen sub licență IBM, unde se descrie extragerea de informații în trei pași. Aceste faze reprezintă o metodologie ce încorporează obiective de domeniu și business, și surse de informație. Fazele sunt explorare, înțelegere și analiză, fiecare dintre ele exprimând capabilități care decurg din cea anterioară.

Faza de explorare constă din parsarea unui volum mare de informații dintre care o parte nu este relevantă pentru rezolvarea problemei propuse. Se localizează astfel mulțimea relevantă a datelor dintr-un set mult mai mare. Pentru date structurate, explorarea se poate face cu interogări, însă dacă se iau în considerare date nestructurate, se utilizează metode de căutare combinate cu operații pe mulțimi. Căutarea este procesul prin care se parsează documente pentru a găsi cuvinte sau fraze specifice în textul nestructurat conținut de acestea. Operațiile pe mulțimi sunt utilizate deoarece de cele mai multe ori o interogare sau căutare nu este de ajuns pentru a obține colecția optimă. Cele mai utilizate operații în aceste cazuri sunt reuniune și intersecție. În unele cazuri, dacă rezultatul combinației de interogări și căutări este prea mare, se aplică tehnici de sampling pentru a obține o submulțime. Aceste tehnici de explorare suplimentare sunt recursiunea interogărilor și expansiunea datelor.

Faza de înțelegere are ca intrare colecția de informații rezultate din faza de explorare și scopul ei este de a descoperi ceea ce informația conține. Se folosește o metodă pentru crearea de structuri din informație nestructurată ce presupune generarea de

taxonomie și rafinare. Procesul de înțelegere este aplicat în două direcții: metodele de analiză încearcă să înțeleagă structura de bază din datele de intrare, iar metodele de date capturează și reprezintă rezultatul.

Se utilizează mai multe mecanisme pentru înțelegerea informației. Partiționarea unui text în secțiuni sau entități mai mici se face pentru ca un text să fie mai ușor de sumarizat și să poată fi analizat mai eficient. Selectarea de caracteristici se aplică după ce o partiționarea este executată cu o granularitate potrivită și constă din analiza statistică a numărului de apariții ale cuvintelor, secvențelor de cuvinte sau frazelor semnificative. Clustering-ul este reprezentat de algoritmi ce grupează documentele în categorii tematice, ce constituie taxonomia. Taxonomia rezultată din clustering este mai apoi editată pentru a include variații de limbaj și nuanțe ale domeniului.

La finalul fazei de înțelegere, se realizează analiza informației structurate și nestructurate pentru a găsi modele, șabloane, corelații, clasificări și relații inerente între date care vor ajuta la decizii de business mai bune.

Data mining reprezintă un real ajutor în întregul proces de descoperire al cunoștințelor, atât la nivel personal cât și în numeroase domenii: medicale, bancare, sociale, afaceri etc. Întregul proces poate duce la luarea unor decizii mult mai rapide, mai eficiente și mai corecte pe baza analizei datelor și extragerea acelor informații utile domeniului în care este utilizat, poate duce la creșterea competitivității, a inovației și a progresului într-un timp mai scurt.

3.2.2. *Frequent Pattern Mining*

Mulțimile alcătuite din elemente care apar frecvent împreună (frequent itemsets) joacă un rol esențial în multe dintre task-urile (sarcinile) de data mining care încearcă să găsească pattern-uri interesante și relevante din cadrul bazelor de date. Aceste pattern-uri pot fi reguli de asociere, corelări, secvențe, clustere, însă regulile de asociere reprezintă unul dintre cele mai tratate subiecte. Motivația inițială de a căuta reguli de asociere a apărut în urma nevoii de analizare a așa-numitelor supermarket transaction data (date ale tranzacțiilor din supermarketuri), care au ca scop examinarea comportamentului cumpărătorilor în ceea ce privește produsele cumpărate. Regulile de asociere, în acest caz, descriu cât de des sunt anumite produse cumpărate împreună. De exemplu, o regulă de formă bere => chipsuri (80%) reprezintă faptul că 4 din 5 clienți care au cumpărat la un moment dat bere, au cumpărat și chipsuri odată cu ea. Astfel de reguli pot fi utile atunci când vine vorba de luarea de decizii cu privire la stabilirea prețurilor, reducerilor sau aranjarea obiectelor în magazin.

De la introducerea lor în 1993 de către Argawal, problemele căutărilor de reguli de asociere și de mulțimi de elemente care apar frecvent împreună au primit un mare grad de atenție. În cursul deceniului trecut, au fost publicate sute de articole care prezentau noi algoritmi sau îmbunătățiri asupra celor existenți astfel încât să se rezolve cât mai bine și mai eficient aceste probleme de căutare. Unul dintre cei mai importanți astfel de algoritmi este Apriori, care va fi descris mai pe larg în subsecțiunea următoare.

3.2.3. Algoritmul Apriori

În data mining, învățarea de reguli de asociere, este o metodă populară de cercetare pentru a descoperi diferite relații dintre variabilele unei mari baze de date. Piatetsky-Shapiro descrie analiza și prezintă reguli semnificative descoperite în bazele de date folosind diferite metode. Bazat pe conceptul de reguli semnificative, Agrawal introduce reguli de asociere cu scopul de a analiza baze de date ample conținând tranzacții comerciale ale unor companii mari de vânzări.

În informatica și data mining, Apriori este un algoritm clasic de învățare a regulilor de asociere. Acesta este proiectat să funcționeze pe baze de date care conțin tranzacții (de exemplu: colecții de obiecte cumpărate de clienți sau detalii despre frecvența vizitelor asupra unui site web). Există și alți algoritmi care sunt proiectați pentru identificarea regulilor de asociere în date care nu conțin tranzacții (Winepi sau Minepi) sau care nu au ștampile de timp (DNA sequencing).

Fie $I = \{i_1, i_2, i_3 \dots i_n\}$ o mulțime de elemente (de exemplu, catalogul produselor vândute). Fie $D = \{t_1, t_2, t_3 \dots t_n\}$ o mulțime de tranzacții comerciale cu aceste elemente, numită "baza de date". Fiecare tranzacție T este o submulțime de elemente, $T \subseteq I$. Tranzacțiile sunt, de obicei, descrise de un identificator de tranzacție, Transaction Id(TxID).

O regulă de asociere este o propoziție logică de forma $A \rightarrow B$, unde $A \cap B = \emptyset$ iar $A, B \subseteq I$. A este antecedentul, iar B consecința regulii. Atât A cât și B sunt grupuri de elemente, submulțimi ale lui I (itemset). Un grup de elemente conținând k elemente este numit k -itemset.

Descoperirea regulilor de asociere este aplicată frecvent, în industrie, peste baze de date conținând un număr mare de tranzacții comerciale. O aplicație uzuală a algoritmilor de detectare de reguli este și problema coșului de cumpărături (market basket analysis), analizarea obiceiurilor de cumpărare prin descoperirea asocierilor de produse selectate frecvent de clienți (deci care apar des împreună în tranzacții, fiecare tranzacție reprezentând un coș de cumpărături).

De exemplu, regula $\{\text{ceapă, cartofi}\} \Rightarrow \{\text{burger}\}$ găsită în datele de vânzare ale unui supermarket ar indica faptul că, dacă un client cumpără ceapă și cartofi împreună, este foarte probabil ca el sau ea să cumpere, de asemenea, și burger. Astfel de informații pot fi utilizate ca bază pentru deciziile cu privire la activitățile de marketing, cum ar fi, stabilirea prețurilor promoționale sau plasarea produselor pe rafturi.

Un alt exemplu din domeniul supermarket-urilor ar fi următorul: dacă am considera mulțimea de elemente $I = \{\text{lapte, pâine, unt, bere}\}$, un exemplu de regulă pentru supermarket ar putea fi: $\{\text{lapte, pâine}\} \Rightarrow \{\text{unt}\}$, ceea ce ar însemna că dacă un client cumpără lapte și pâine, atunci acesta ar cumpără și unt. Trebuie să menționăm faptul că acest exemplu este unul extrem de simplist. În practică, o regulă trebuie să suporte sute de tranzacții până când poate fi considerată semnificativă din punct de vedere statistic, iar mulțimile de date adesea conțin mii sau milioane de tranzacții. Pentru a selecta reguli interesante și semnificative din mulțimea tuturor regulilor posibile, se folosesc o serie de constrângeri asupra anumitor măsurători de importanță sau interes pe care le realizează algoritmul. Cele mai cunoscute dintre acestea sunt limita minimă a suportului sau încrederea (confidence).

3.2.4. Alți algoritmi

Această secțiune va prezenta doi dintre cei mai reprezentativi algoritmi de frequent itemset mining : SaM și Carpenter.

- Algoritmul SaM (Split and Merge)

SaM este un program cu ajutorul căruia se pot găsi mulțimi de elemente frecvente și care folosește algoritmul cu același nume în acest scop. Algoritmul a fost inițial creat în scop educațional, întrucât demonstrează că o căutare în adâncime (Depth First Search) pe “subset lattice” poate fi combinată cu o reprezentare pur-orizentală a tranzacțiilor bazei de date. În acest fel, vine în contrast cu alți algoritmi precum Eclat care tot așa face o căutare în adâncime dar se bazează pe o reprezentare pur verticală a tranzacțiilor din baza de date. SaM funcționează prin însumarea costurilor tranzacțiilor în pasul de “Split”.

Schemă algoritmică de bază a fost descrisă inițial în Borgelt and Wang – 2009 , care acoperă de asemenea și modul în care se pot găsi mulțimi aproximative de elemente frecvente cu ajutorul lui SaM. Ideea principală este că tranzacțiile din baza de date reprezintă observații nu tocmai exacte ale tranzacțiilor reale, cu alte cuvinte o tranzacție poate să conțină un element chiar dacă acesta nu a fost înregistrat în baza de date. Pentru a gestiona o astfel de situație, SaM permite unei mulțimi de elemente să fie suportate de tranzacții care conțin doar părți ale acelor elemente, chiar dacă aceste tranzacții au un cost mai mic. Costul unei tranzacții este calculat în funcție de ce elemente mai trebuie inserate în aceasta astfel încât tranzacția să conțină o mulțime de elemente frecvente dintre cele luate în considerare. Astfel, cu câte mai multe elemente trebuie inserate, cu atât este costul mic. Principalul avantaj al acestui algoritm nu este neapărat viteza sa, chiar dacă stă bine și la acest capitol, ci mai degrabă simplitatea structurii sale. Practic, toată muncă este realizată într-o funcție recursivă de câteva linii de cod, iar singura structură de date folosită este un vector simplu de elemente.

- Algoritmul Carpenter

Scheema de bază a acestui algoritm a fost dezvoltată în Pan – 2003, și funcționează prin enumerarea de mulțimi de tranzacții, în contrast cu mulți alți algoritmi de frequent itemset mining care enumeră mulțimi de elemente. O asemenea abordare poate fi foarte eficientă în anumite cazuri, în special atunci când avem puține tranzacții și foarte multe elemente, situație adesea întâlnită în cadrul mulțimilor de date biologice cum ar fi cele de exprimare a datelor genetice. Pentru alte cazuri în care există mai puține elemente și mai multe tranzacții, o asemenea abordare nu este neapărat recomandată.

În mod implicit, algoritmul găsește mulțimi închise de elemente, însă poate găsi și mulțimi maximale. Acesta are două alternative de implementare. Prima se bazează pe ID-urile tranzacțiilor așa cum este descrisă în varianta originală în Pan – 2003, iar cea de-a doua este bazată pe tabelul care ține evidența elementelor apărute. Care dintre cele două variante este mai rapidă depinde de mulțimea de date pe care se aplică, iar alegerea implementării se face automat, ținând cont de dimensiunile acestora – dacă mulțimea de date încapă în cache-ul procesorului atunci se folosește varianta a doua, cu tabelul

elementelor apărute, iar altfel se utilizează prima variantă, bazată pe lista de ID-uri ale tranzacțiilor.

3.3. Procesarea textelor

3.3.1. Procesare texte cu parsare lingvistică

La secțiunea 3.1 am amintit despre “Procesarea limbajului natural”, ce este și cu ce se ocupă. Procesarea limbajului natural oferă una dintre modalitățile de a face interacțiunea om-calculator mai interesantă și mai accesibilă. Determinarea dependențelor sintactice între cuvintele dintr-o frază reprezintă o sarcină importantă în acest domeniu, fiind utilă pentru o varietate de aplicații, printre care traducerea automată, extragerea și clasificarea opiniilor din texte, aplicații de tip întrebare-răspuns și altele.

În cadrul procesării limbajului natural, una dintre cele mai importante acțiuni de efectuat se numește “parsare”. Aceasta se referă, în general, la analizarea de fraze, determinarea relațiilor dintre cuvintele respectivelor fraze și construirea de arbori de legături. Acești arbori de parsare pot fi de două tipuri.

- a) Arbori cu parsare sintactică - aceștia se referă la structura unei fraze, și se poate obține folosind gramatici independente de context sau probabilistice

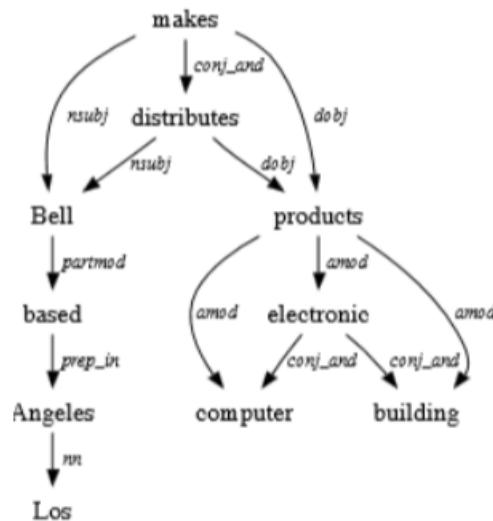


Figura 3.1 Exemplu de arbore de parsare sintactic

- b) Arbori de dependențe - acest tip de arbore arată relațiile gramaticale care pot fi regăsite într-o propoziție, cum ar fi atributele, complementele, sau alte elemente asemănătoare.

```

(ROOT
  (S
    (NP
      (NP (NNP Bell))
      (, ,)
      (VP (VEN based)
        (PP (IN in)
          (NP (NNP Los) (NNP Angeles)))))
      (, ,))
    (VP (VBZ makes)
      (CC and)
      (VP (VBZ distributes)
        (NP
          (UCP (JJ electronic) (, ,) (NN computer)
            (CC and)
            (NN building))
          (NNS products))))
      (. .)))
  )

```

Figura 3.2 Exemplu de arbore de de dependențe

Relațiile dintre cuvinte sunt printre cele mai importante elemente din domeniul procesării limbajului natural. Arborii prezentați mai sus sunt folosiți în multe situații de procesare a limbajului, cum ar fi în cadrul aplicațiilor de traducere automată care folosesc arbori de dependențe. Acest lucru face ca algoritmi de învățare și structura frazei să genereze rezultate mai bune decât în cazul altor modele.

O componentă importantă a procesării limbajului natural este procesul de adnotare cu etichete morfo-sintactice (“Part-of-speech tagging – POS tagging”). Adnotarea cu etichete morfo-sintactice (“POS tagging”) este procesul de etichetare gramaticală a fiecărui cuvânt dintr-o propoziție, frază sau paragraf cu partea de vorbire corespunzătoare. Eticheta fiecărui cuvânt reprezintă partea de vorbire pe care acesta îl are ca rol în contextul propoziției (de exemplu: substantiv, verb, prepoziții, interjecții). Pe lângă părțile de vorbire, cuvintele etichetate pot să conțină informații suplimentare legate de caracteristicile morfologice ale limbii respective, precum număr, gen, persoană, timpul sau aspectul verbului. Ca și metode de rezolvare pentru operația de Part-of-speech tagging, a fost făcută o clasificare în acest sens de către Jurafsky și Martin în anul 2000. Aceștia au împărțit metodele de PoS tagging în trei categorii :

- Metode stocastice
- Metode bazate pe reguli
- Metode hibride

Prima categorie de metode are nevoie de o colecție de texte de pe urma căruia să se antreneze, cu alte cuvinte să învețe de pe urma acestora. Metodele stocastice calculează probabilitatea unei etichete pentru fiecare cuvânt analizat, într-un context predefinit. PoS tagger-ele de acest gen sunt numite probabilistice și necesită un proces de învățare dintr-o cantitate mare de date de antrenament pentru a se ajunge la o precizie ridicată.

Cea de-a doua categorie de metode se bazează în prima fază pe un dicționar care conține cuvinte mapate pe liste de părți de vorbire posibile. Astfel, pentru fiecare cuvânt se păstrează o mulțime de părți de vorbire care îi pot corespunde, în funcție de context. Apoi, aceste metode mai au nevoie și de un set de reguli care să elimine ambiguitățile.

Aceste reguli vor face referire la secvența de părți de vorbire permise într-un anumit context. Un exemplu de astfel de PoS tagger este cel numit “English Constraint Grammar Parser”, prezentat de Voutilainen în 1995, care a avut o precizie de 99.7%.

Nu în ultimul rând, există și o abordare hibridă care combină cele două categorii prezentate mai sus. Un exemplu de acest fel este PoS tagger-ul propus de Brill în 1995, cu o precizie de 96-97%. Acesta folosește reguli deduse din date folosind inițial un set de texte etichetate, de antrenament, pentru a stabili cazurile în care un cuvânt ambiguu poate avea o anumită etichetă.

Procesul de etichetare a cuvintelor unui text este unul relativ complex, care depinde de o serie de factori. Acești factori au fost identificați de către Manning și Schutze în 1999, și sunt prezentați în cele ce urmează:

- Dimensiunea setului de antrenare. Precizia unui PoS tagger depinde în mare măsură de cantitatea de date de antrenament, în așa fel încât cu cât aceste date sunt mai numeroase, cu atât precizia finală este mai mare.
- Setul de etichete folosit. Aici trebuie să existe un echilibru, întrucât un set mare de etichete poate să inducă ambiguitate, însă permite obținerea mai multor trăsături morfologice ale unui cuvânt într-un context dat.
- Legătura dintre datele de antrenament și datele de testare. Precizia va fi cu atât mai ridicată cu câte aceste două seturi de date sunt mai asemănătoare, de exemplu din același domeniu, și dacă ele sunt scrise de-a lungul aceleiași perioade de timp. Dacă folosim de exemplu un set de texte științifice și unul din ziare, precizia va scădea semnificativ.
- Cuvintele necunoscute. Acestea se referă la cuvintele care nu se găsesc în setul de antrenament, dar pe care PoS tagger-ul trebuie totuși să le eticheteze. Numărul de cuvinte necunoscute crește mai ales în situațiile în care textul pe care se aplică PoS tagger-ul este din alt domeniu față de textele de învățare, sau conțin particularități, cum ar fi de exemplu conversațiile.

Precizia Precizia PoS tagger-elor a fost îmbunătățită de-a lungul timpilor prin apariția unor diferiți algoritmi, în special care să se refere la cazul cuvintelor necunoscute, folosind diferite trăsături ale cuvântului sau contextului în care se află acesta. Jurafsky și Martin în lucrarea din anul 2000 prezintă o serie de astfel de algoritmi. Unul dintre ei a fost propus de Baayen și Sproat în 1996 și consideră că, pentru cuvintele necunoscute, distribuția probabilă de etichete este similară cu distribuția etichetelor pentru cuvintele care apar o singură dată într-un set de antrenament, adică acelea care au o singură parte de vorbire asociată. Wischedel în lucrarea sa din 1993 a descris un alt algoritm mai eficient pentru găsirea etichetei unui cuvânt necunoscut. Acesta se bazează pe regulile ortografice și ia în considerare următoarele trăsături : morfemele flexionare, morfemele derivative, scrierea cu majuscule și despărțirea în silabe.

3.3.2. *Procesare texte fără parsare lingvistică*

În procesul de extragere de informații din date și în special din texte în limbaj natural, o bună parte dintre sistemele proiectate în acest scop se bazează pe tehnici de machine learning care au nevoie de o parsare lingvistică a documentelor. Totuși, alte câteva abordări au încercat să evite această parsare lingvistică și să se bazeze mai mult pe

găsirea de reguli de asociere și construirea unor arbori de decizie , în așa fel încât nu mai este nevoie de parsarea lingvistică a textului.

Aceste abordări nu au fost însă dezvoltate foarte mult, probabil și din cauza faptului că încă de la început majoritatea abordărilor implicau parsarea lingvistică inițială a textului, iar cei care începeau să studieze acest domeniu se axau mai degrabă pe îmbunătățirea sistemelor existente și pe abordări similare, în locul abordării de tehnici noi. Totuși, în ultima perioadă au început să apară și variante diferite care să evite parsarea lingvistică și să se concentreze asupra utilizării de reguli de asociere pentru a extrage cu ajutorul lor datele din informații nestructurate.

3.4. Alte studii și realizări

Dintre articolele și lucrările studiate în procesul de realizare a acestei lucrări, unul dintre cele mai relevante am considerat că ar fi articolul [1] , cu titlul “Extracting medical information from narrative patient records: the case of medication-related information”, pe care îl voi descrie pe scurt în cele ce urmează.

Lucrarea menționată mai sus abordează o metodă de extragere automată a informațiilor legate de medicație, din seturi de date clinice, precum și o serie de strategii testate pentru a îmbunătăți procesul de extragere. Abordarea se bazează pe extragerea de reguli în două etape: prima etapă constă în recunoașterea numelor de medicamente, iar cea de-a doua etapă constă în explorarea contextului acestor nume pentru a extrage informații în legătură cu medicamentele (mod de administrare, doze, etc). acest lucru se realizează pe baza unor reguli, capturând structura documentului și sintaxa fiecărei informații. Au fost testate mai multe configurații cu scopul de a îmbunătăți performanța sistemului, în particular procesul de recunoaștere a numelor de medicamente, acesta fiind unul dintre cei mai importanți factori din procesul de extragere.

Sistemul inițial a avut rezultate bune, cu o precizie în jurul valorii de 77%, iar în urmă testării diferitor configurații s-a ajuns ulterior la 81%. Cele mai precise detecții s-au făcut asupra numelor de medicamente și modului de administrare(84% respectiv 88%), însă atunci când a venit vorba de durata administrării și motive, situația a fost puțin mai problematică.

Concluzia acestui studiu a fost faptul că un simplu sistem bazat pe reguli poate să obțină rezultate foarte bune în cazul extragerii de informații despre medicamente, iar în urmă unor modificări controlate se pot obține îmbunătățiri semnificative.

Capitolul 4. Analiză și Fundamentare Teoretică

4.1. Arhitectura conceptuală a sistemului

În proiectarea acestui sistem am folosit o serie de bune reguli și practice din programarea orientată obiect, astfel am construit arhitectura sistemului pe baza unui pattern arhitectural, și anume Model-View-Controller. Rolul principal al acestui pattern arhitectural este separarea logicii interne a unei aplicații de partea sa de prezentare, oferind posibilitatea modificării independente a acestor componente.

Structura, proprietățile și avantajele acestui pattern arhitectural vor fi prezentate în detaliu în secțiunea următoare.

4.1.1. Pattern-ul arhitectural Model-View-Controller

Model-View-Controller (MVC) este un pattern arhitectural utilizat în ingineria software pentru o mai bună dezvoltare a aplicațiilor. Succesul acestui pattern se datorează faptului că permite izolarea logicii interne a unei aplicații de partea sa de prezentare (interfață), prin împărțirea responsabilităților. Organizarea MVC, oferă posibilitatea modificării independente a componentelor, fără a afecta alte nivele și facilitează reutilizarea acestora. După cum sugerează și numele acestuia, MVC descompune aplicația în trei părți: partea de model, view și controller.

Arhitectura acestui pattern este următoarea :

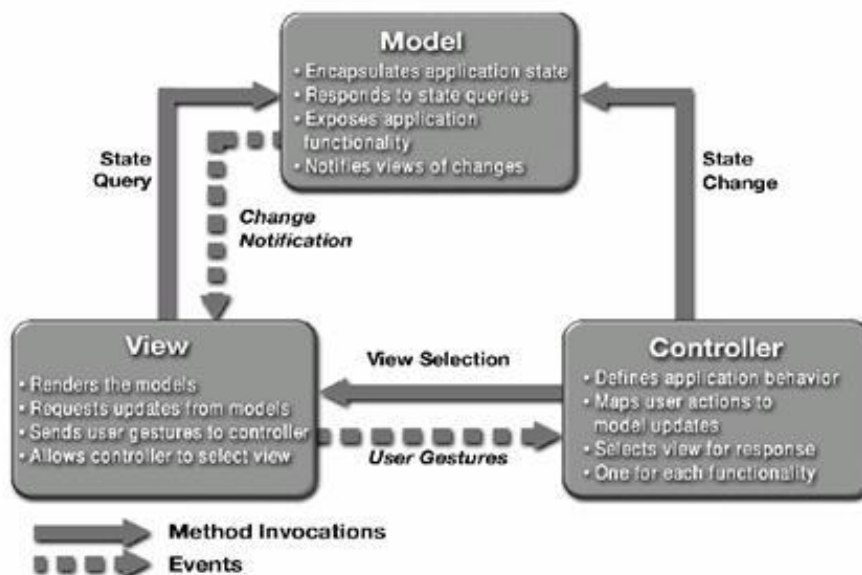


Figura 4.1 Arhitectura design pattern-ului Model-View-Controller

Model – menține starea curentă a aplicației, a unei părți a aplicației sau a unui set de date, putând de asemenea executa anumite acțiuni. Această componentă nu are nici o informație cu privire la modul în care informațiile vor fi afișate către utilizator. Această

componentă trebuie să ofere metode prin intermediul cărora modelul poate fi interogată cu privire la starea curentă.

View – prezentarea datelor către utilizator, de obicei sub formă unor componente grafice care reflectă starea curentă a unui model. Pot exista mai multe view-uri asociate cu un același model.

Controller – procesează și răspunde la evenimente. În mod uzual controllerul preia acțiunile utilizatorului și acționează asupra componentei Model sau asupra componentei View. Fiecare View are asociat câte un controller care conține câte un listener pentru fiecare buton din View. Acesta are acces atât la model, cât și la view-ul său și are rolul de a crea o legătură între acestea.

4.1.2. Prezentarea arhitecturii conceptuale

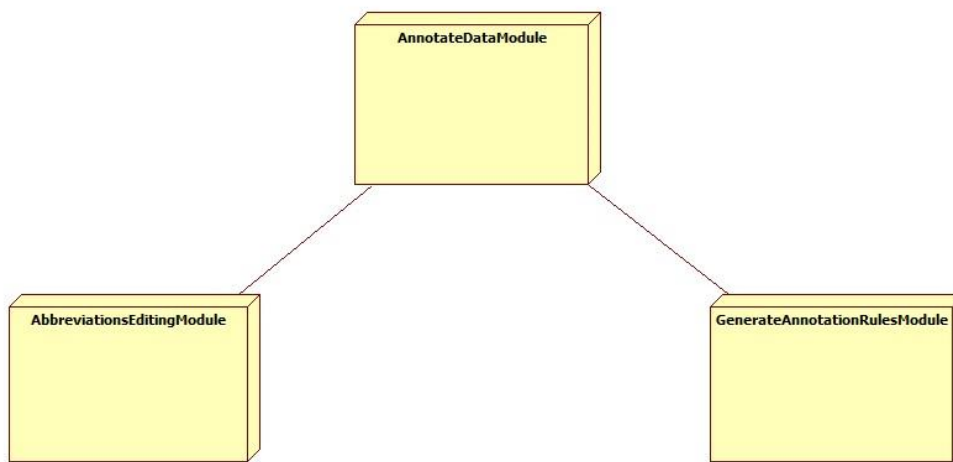


Figura 4.1 Diagrama conceptuală a sistemului

În figura 4.1 este prezentată diagrama conceptuală al acestui sistem. Este formată din trei module principale:

1. Modulul de adăugare și actualizare al abrevierilor fiind reprezentat de AbbreviationsEditingModule
2. Modulul de învățare și generare de reguli fiind reprezentat de GenerateAnnotationRulesModule
3. Modulul de adnotare a datelor fiind reprezentat de AnnotateDataModule

Fiecare dintre aceste trei module, are rolul său unic în cadrul aplicației, funcționalitatea acestora depinzând în mare parte de fiecare dintre ele.

În continuare vor fi prezentate mai în detaliu modulele componente ale acestei arhitecturi.

4.1.3. Modulul de procesare al abrevierilor

După cum am descris și în partea introductivă a acestei lucrări, obiectivul nostru principal este realizarea unei aplicații utile în domeniul medical. Acest domeniu este

foarte vast în termeni de informație, cuprinzând o multitudine de cuvinte cheie, definiții sau termeni specifici. La scrierea acestor informații, specialiștii folosesc adesea cuvinte prescurtate, notații ambigue și o mulțime de alte abrevieri. În mod normal, un specialist știe să citească și să descifreze astfel de noțiuni, dar situația nu este la fel atunci când vine vorba de procesarea informațiilor de către un calculator.

Astfel, la analiza textelor din domeniul medical, primul pas este să se găsească o soluție la această mică problemă. Din moment ce este oricum destul de dificil procesarea unui text scris în limbaj natural, întâmpinarea acestor prescurtări ne complică și mai mult muncă. În scopul rezolvării acestei probleme, soluția propusă constă în crearea unui dicționar în care se vor păstra aceste abrevieri. Acesta va fi structurat în stilul clasic, conținând înregistrări de tip abreviere – semnificație. Pentru fiecare prescurtare sau notație care poate fi întâmpinată în cadrul setului de date cu care lucrăm, indiferent că este vorba de datele de învățare sau cele care vor fi prelucrate, va exista o înregistrare în acest dicționar care să conțină prescurtarea respectivă împreună cu semnificația acesteia. Scopul principal al acestei structuri prezentate este folosirea ulterioară a sa în etapa de analiză a observațiilor medicale. În acest fel, înlocuind în cadrul observațiilor fiecare abreviere cu semnificația sa, textul va primi un înțeles mai clar, ambiguitatea va fi eliminată iar procesarea va fi efectuată într-un mod mai corect.

Având în vedere că această clarificare a textelor este un pas important în procesul de prelucrare al informației, am ales construirea unui modul special dedicat acestei operații. Acesta este modulul de procesare al abrevierilor, care se învârte în jurul dicționarului de care aminteam mai sus, oferind asupra lui o serie de operații care sunt neapărat necesare. Printre aceste operații se numără:

- Adăugarea unei înregistrări noi, abreviere – semnificație
- Ștergerea unei înregistrări existente
- Salvarea modificărilor efectuate
- Căutarea după o anumită abreviere

Pentru a oferi o vedere mai bună a ceea ce se va putea face cu ajutorul acestui modul, voi prezenta în continuare o diagramă a cazurilor de utilizare. Actorul acestui caz este utilizatorul aplicației, și anume expertul uman.

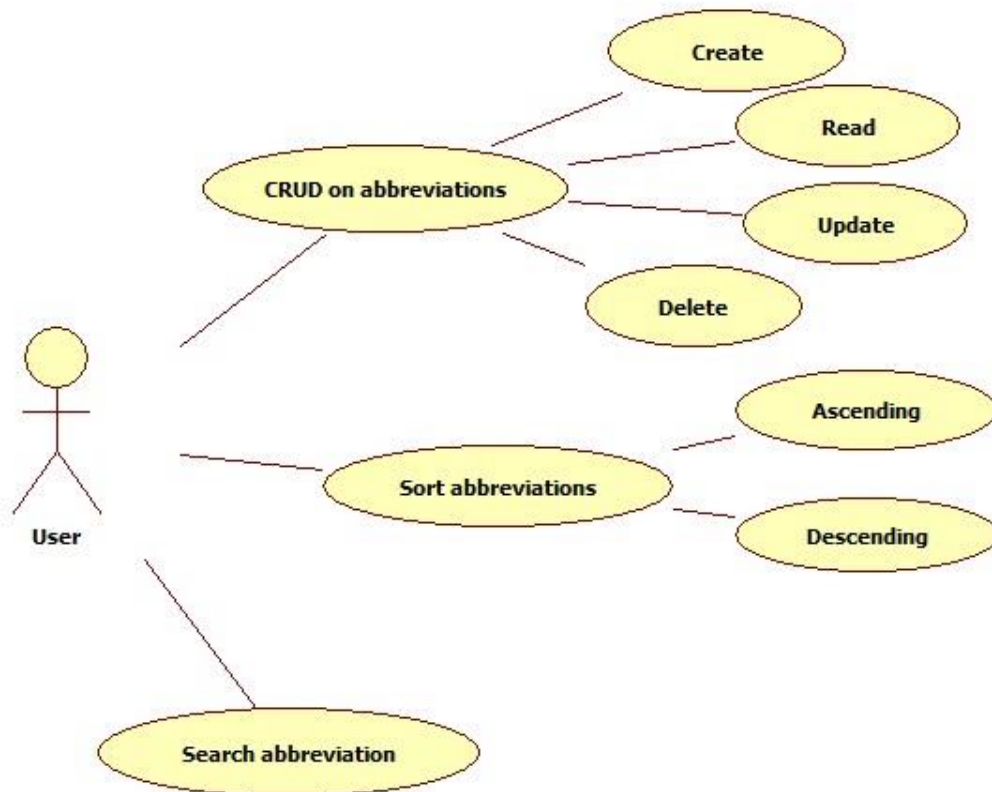


Figura 4.2 Diagrama use-case pentru modulul de abrevieri

Pentru o mai bună și mai ușoară utilizare a acestui modul, și pentru ușurință realizării operațiilor descrise mai sus, am creat o interfață grafică special dedicată acestui modul. Din cadrul acesteia, utilizatorul va putea efectua întregul set de operații amintite cu ușurință. Modul în care se realizează aceste operații va fi descris mai pe larg în secțiunea corespunzătoare din capitolul care face referire la implementarea sistemului.

4.1.4. Modulul de generare de reguli

Aplicațiile din acest domeniu de procesare a limbajului natural, așa cum aminteam și în secțiunile anterioare, au nevoie de o etapă de învățare pentru că mai apoi să poată opera cu succes și într-un mod cât mai performant asupra seturilor de date. Etapa de învățare presupune în general crearea de reguli de pe urma unor mulțimi de date, care mai apoi să poată fi aplicate pe alte seturi de informații cu scopul de a le procesa și de a le înțelege. Astfel, și în cadrul aplicației descrise în această lucrare e nevoie de o asemenea etapă de învățare, iar în scopul realizării acesteia am creat acest modul special dedicat care va fi descris în această secțiune. Pentru început, este prezentată diagrama cazurilor de utilizare existente pentru un utilizator.

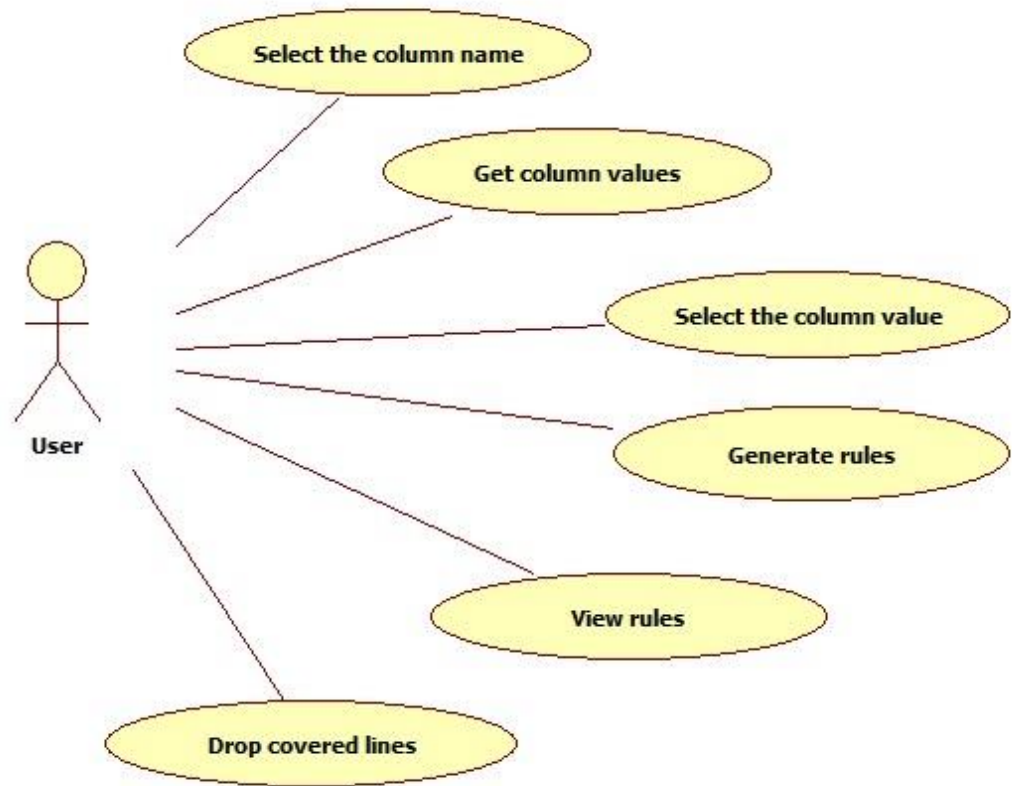


Figura 4.3 Digrama use-case pentru modulul de generare de reguli

Scopul principal al acestui modul este de a examina anumite seturi de date deja procesate și adnotate, pentru a genera reguli pe baza acestora. Regulile vor fi necesare în următoarea etapă, cea de adnotare a observațiilor medicale. Modulul oferă posibilitatea ca utilizatorul să aleagă setul inițial de date și să obțină pentru acestea o primă mulțime de reguli. În acest prim pas, regulile sunt generate în urma operației de găsire a pattern-urilor frecvente care apar în observațiile analizate și adnotate, și au următoarea structură:

ElemDiagnosticat_adnotare <- diagnoza

Exemple:

- S-Lum_RDG <- duodeno gastric
- S-Lum_DIL <- stomac dilatat

Acest proces de generare se va realiza cu ajutorul algoritmului Apriori. Modul în care acesta operează asupra setului de date medicale în contextul acestei aplicații va fi prezentat în secțiunea 4.4. Pentru moment, trebuie să menționăm faptul că fiecare regulă astfel generată are o anumită valoare numită “support”. Rolul acestei valori este de a indica numărul total de apariții ale unui grup de cuvinte care apar frecvent împreună, raportat la numărul total de observații din mulțimea de date examinate. Așadar, cu cât support-ul este mai mare, cu atât acel grup de cuvinte – în cazul nostru acea regulă – acoperă o mai mare parte din observațiile studiate. Asta înseamnă că respectivă regulă va fi mai precisă iar la utilizarea ei ulterioară va avea șanse semnificativ mai mari să genereze adnotări corecte.

Pe lângă acest prim pas de generare inițială a regulilor, utilizatorul are posibilitatea de a rafina acest set inițial. Această operație se realizează prin eliminarea regulilor pe care expertul uman le consideră mai puțin revelante sau nu foarte precise, și memorarea acelor pe care le considera utile în cadrul procesului de adnotare. După selectarea regulilor dorite și memorarea acestora pentru utilizarea ulterioară, utilizatorul mai are posibilitatea de a executa încă o dată generarea de reguli, de dată această pe datele inițiale din care au fost eliminate înregistrările care conțineau seturile de cuvinte din partea dreaptă a regulilor selectate. În urma acestei operații, mulțimea de reguli se va restrânge iar noi valori pentru support vor fi generate.

4.1.5. Modulul de adnotare

Una dintre cele mai importante responsabilități ale aplicației prezentate în această lucrare este etichetarea observațiilor medicale scrise în limbaj natural. Întrucât aceste observații sunt scrise adesea în formă prescurtată, folosind notații și adnotări specifice domeniului medical, etapa de adnotare trebuie să fie executată abia după ce observațiile sunt transformate într-o formă mai explicită. De asemenea, adnotarea se va efectua pe baza regulilor generate în cadrul etapei de învățare, descrisă în secțiunea anterioară. Similar cu celelalte două etape prezentate deja, și aceasta are un modul dedicat cu ajutorul căruia se pot efectua operațiile necesare.

Procesul de etichetare a observațiilor medicale poate fi împărțit în două etape, fiecare dintre acestea având un rol important în obținerea de rezultate cât mai consistente și mai corecte. Așadar, prima etapă este reprezentată de transformarea observațiilor din formă prescurtată și ambiguă în formă completă, iar cea de-a doua etapă constă în procesul de generare propriu-zisă a etichetelor pentru aceste observații. La fel ca și în cazurile anterioare, utilizatorul poate efectua aceste operații prin intermediul unei interfețe grafice, diagrama cazurilor de utilizare fiind prezentată mai jos:

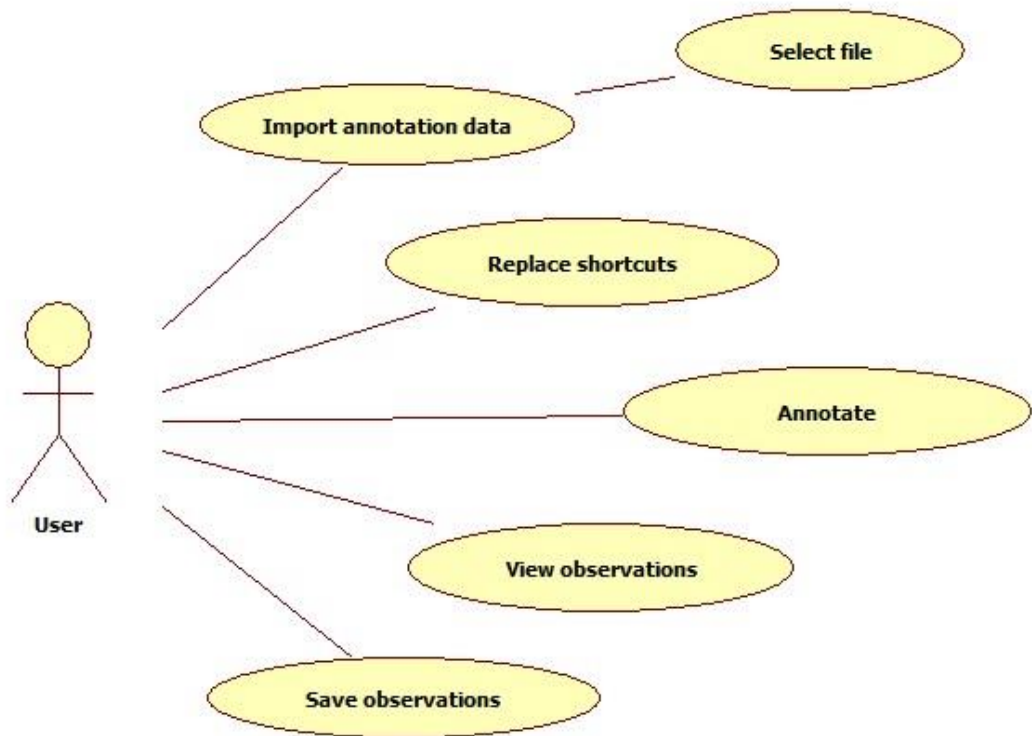


Figura 4.3 Diagrama use-case pentru modulul de adnotare

Prima etapă, cea de parcurgere și prelucrare a observațiilor are loc prin procesul de înlocuire al abrevierilor. Acest proces se realizează pe baza dicționarului de abrevieri creat în prealabil, înlocuind în fiecare observație medicală abrevierile întâlnite cu semnificațiile lor din cadrul dicționarului. Apoi, aceste observații vor fi pregătite pentru etapa a doua, cea de parsare și etichetare.

Această a doua etapă se realizează prin parcurgerea setului de date cu abrevierile înlocuite și aplicarea regulilor de asociere corespunzătoare asupra acestora. Asta înseamnă că acest pas trebuie să vină în urma celui de generare a regulilor de asociere, despre care s-a vorbit în subcapitolul 4.1.2. După aplicarea acestor reguli, se vor genera etichete specifice fiecărei observații în parte, în funcție de potrivirile care au loc.

4.2. Modelul de date

4.2.1. Prezentare generală

Fiecare sistem software are la dispoziție un model de date asupra căruia are loc întregul procesul aplicației. Acest model are un rol foarte important în cadrul unei aplicații deoarece stă la baza acesteia. În primul rând, majoritatea sistemelor operează asupra unor date de intrare care urmează să fie procesate în interiorul acestora. După procesele care se execută, are loc generarea datelor de ieșire, care reprezintă rezultatele

operațiilor efectuate de către aplicație. O constrângere importantă este că modelul de date utilizat de aplicație trebuie să fie structurat în așa fel încât aceasta să îl poată procesa.

Un model de date reprezintă o abstractizare a anumitor informații din lumea reală, și reprezentarea lor pe sistemele de calcul. Acestea pot fi structurate într-o mare varietate de formate, și pot fi utilizate de diferite categorii de sisteme și aplicații. Un model particular de date ar putea să permită existența simultană a mai multor reprezentări ale informației, sau pe de altă parte ar putea să încerce să dezvolte o reprezentare comună a acestor formate. De asemenea, modelele se pot împărți în prima fază în două categorii : interne – care sunt destinate implementării în cadrul aplicației , sau externe – care sunt utilizate și înțelese de către utilizatorii acestor aplicații.

În general, un astfel de model de date specifică modul de organizare și definire al informațiilor. Acesta poate să facă referire nu doar la structura fizică a obiectelor de date și implementarea acestora, ci și structura logică a respectivelor date. Prin structura logică înțelegem valorile memorate împreună cu proprietățile logice ale acestora, dar și modul în care anumite obiecte mai complexe sunt alcătuite din altele mai simple. De asemenea, comportamentul datelor în cadrul proceselor de prelucrare ale aplicației este din nou un lucru pe care modelul de date trebuie să îl descrie.

4.2.2. Datele procesul de învățare

Așa cum am prezentat în secțiunea de descriere a a arhitecturii sistemului, aplicația din cadrul acestei lucrări are nevoie să treacă printr-o etapă de învățare. Ca și orice altă operație sau etapă din procesul de prelucrare, această învățare necesită o serie de date de intrare asupra cărora să se opereze. După ce acestea vor fi trecute printr-o serie de operații și procese, se vor obține date de ieșire care sunt necesare etapei următoare, și anume etapa de adnotare a observațiilor. În continuare vom prezenta pe scurt structura acestor date de intrare.

Informațiile de care procesul de învățare are nevoie în primul rând sunt reprezentate de o mulțime de observații medicale similare cu cele asupra cărora aplicația va opera ulterior și va realiza procesul de adnotare. Aceste observații trebuie să fie scrise în prealabil de experți umani și trebuie etichetate manual.

Un exemplu concret de astfel de observații etichetate, din setul de date de intrare, este prezentat în tabelul de mai jos:

Tabel 4.1 Exemplu de observatii etichetate manual

Stomac	S-Lum	S-Cont	S-MucANT	S-MucJAC	S-MucCORP
Hemoragic	N	N	N	N	N
Partial volvulat	VGI	N	N	N	N
Staza gastrica cu mult suc gastric si bila congestionat la nivelul antrului	N	BI_SU	ER	N	N

În urma prelucrării acestor date, se vor obține informații noi prin generarea de reguli, iar scopul acestora este de a fi utilizate ca și date de intrare de către următoarea etapă.

4.2.3. Datele procesul de adnotare

Procesul de adnotare este ultima etapă din cadrul acestei aplicații și am putea spune că este cea mai semnificativă. Asemenea ca și la procesul de învățare, această etapă necesită un set de date asupra cărora să se efectueze operațiile. După ce datele inițiale au fost trecute printr-o serie de operații la procesul de învățare, datele de ieșire obținute, vor servi ca date de intrare la etichetarea observațiilor.

Prima operație care trebuie efectuată în acest proces este cea de înlocuire a abrevierilor din textele medicale. După obținerea observațiilor de către medici în urma analizei de endoscopie, acestea trebuie prelucrate deoarece conțin o serie de abrevieri și prescurtati ambigue. Așa cum am menționat și în secțiunea 4.1.3 acesta operație se realizează pe baza dicționarului de abrevieri, înlocuind fiecare abreviere întâlnită cu semnificația ei din cadrul dicționarului. În urma operației de expandare a abrevierilor se vor obține noi informații care vor fi utile la pasul următor al acestui proces.

A doua operație care urmează a fi efectuată este cea de adnotare a textelor medicale. Această operație are nevoie ca și date de intrare rezultatele obținute la operația de etichetare și rezultatele obținute la procesul de învățare. Pe baza acestor informații, operația de adnotare va putea fi efectuată cu succes.

4.3. Algoritmul Apriori

Noțiunile de bază și rolul principal al algoritmului Apriori au fost prezentate în secțiune 3.2.3. În continuare în cadrul acest subcapitol vor fi prezentate pseudocodul algoritmului, modul acestuia de funcționare și principalii parametri cu care operează.

În cele ce urmează voi prezenta principali parametri ai algoritmului Apriori și îi voi exemplifica pe baza unui mic set de date de test. Pentru aceasta mă voi axa pe un mic exemplu din domeniul supermarket.

Se dă setul de elemnete $I = \{\text{lapte, pâine, unt, bere}\}$ și o bază de date mică care conține elementele (numărul 1 indică prezența unui element dintr-o tranzacție iar 0 absența acestuia) descrisă în tabelul de mai jos. Un exemplu de regulă de supermarket ar putea fi : $\{\text{lapte, pâine}\} \rightarrow \{\text{butter}\}$ ceea ce înseamnă că dacă laptele și pâinea sunt cumpărate împreună, clienții ar cumpără de asemenea și unt.

Tabel 4.2 Exemplu de tranzacții

ID Transactie	Lapte	Paine	Unt	Bere
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0
6	1	0	0	0
7	0	1	1	1

8	1	1	1	1
9	0	1	0	1
10	1	1	0	0
11	1	0	0	0
12	0	0	0	1
13	1	1	1	0
14	1	0	1	0
15	1	1	1	1

Pentru a selecta reguli interesante și relevante din mulțimea totală de reguli posibile, se pot folosi o serie de constrângeri asupra diferitelor măsurători de importanță și interes. Două dintre cele mai importante astfel de constrângeri sunt valoarea minimă pentru suport și încrederea (confidence).

Support

Supportul $\text{supp}(X)$ al unei mulțimi de elemente X este definit ca fiind raportul dintre numărul de tranzacții în care apare această mulțime și numărul total de tranzacții.

$$\text{supp}(X) = (\text{numărul de tranzacții care conțin mulțimea } X) / (\text{numărul total de tranzacții})$$

În datele de test prezentate mai sus, mulțimea {lapte, pâine, unt} are un suport de $4/15 = 0.26$, ceea ce înseamnă că apare în 26% dintre tranzacții.

Confidence

Încrederea unei reguli este definită ca:

$$\text{conf}(X \rightarrow Y) = \text{supp}(X \cup Y) / \text{supp}(X)$$

Pentru regula {lapte, pâine} \Rightarrow {unt}, avem următoarea încredere:

$$\text{Supp}(\{\text{lapte, pâine, unt}\}) / \text{supp}(\{\text{lapte, pâine}\}) = 0.26 / 0.4 = 0.65.$$

Asta înseamnă că pentru 65% din tranzacțiile care conțin lapte și pâine, regula este corectă, adică respectivele tranzacții conțin și unt. Încrederea poate fi interpretată ca o estimare a probabilității $P(Y | X)$, și anume probabilitatea ca partea dreaptă a regulii să se găsească în tranzacțiile care conțin partea stângă a regulii.

Lift

Aceasta aparține de o regulă, și este definit de următoarea formulă:

$$\text{lift}(X \rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(Y) * \text{supp}(X)}$$

Regula {lapte, pâine} \Rightarrow {unt} are următorul lift:

$$\text{Supp}(\{\text{lapte, pâine, unt}\}) / \text{supp}(\{\text{lapte, pâine}\}) = 0.26 / 0.46 * 0.4 = 1.4$$

Conviction

Gradul de convingere al unei reguli este dat de următoarea formulă:

$$conv(X \rightarrow Y) = \frac{1 - supp(Y)}{1 - conf(X \rightarrow Y)}$$

Regula {lapte, pâine} => {unt} are următoarea convingere:

$$1 - supp\{unt\} / 1 - conf(\{lapte, pâine\}) => \{unt\} = 1 - 0.46 / 1 - 0.65 = 1.54.$$

Convingerea unei reguli $X \Rightarrow Y$ poate fi interpretată ca fiind măsura în care se așteaptă ca X să apară fără Y (adică frecvența cu care regula face o prezicere incorectă) în cazul în care X și Y ar fi independente. Pe acest exemplu, o convingere de 1.54 reprezintă faptul că regula {lapte, pâine} => {unt} ar fi incorectă cu 54% mai mult dacă asocierea dintre X și Y ar fi făcută aleator.

Generarea regulilor de asociere este, de obicei împărțită în doua etape :

1. Prima etapă constă în aplicarea suportului pentru a găsi toate mulțimile de elemente frecvente dintr-o bază de date.
2. Aceste mulțimi de elemente frecvente generate la pasul anterior, împreună cu constrângerea de încredere minimă sunt utilizate pentru a forma reguli.

În timp ce al doilea pas este mai simplu și mai direct, primul pas necesită mai multă atenție. Găsirea tuturor mulțimilor de elemente frecvente din baza de date este dificilă, deoarece această implică căutarea tuturor mulțimilor de elemente posibile (combinații de elemente). Dacă am grupa aceste mulțimi de elemente posibile într-o singură mulțime, aceasta ar fi de dimensiunea $2^n - 1$ (făcând abstracție de mulțimea vidă, care defapt nu este o mulțime de elemente). Chiar dacă dimensiunea acestei mulțimi (numită “power set”) crește exponențial dacă ne raportăm la numărul “ n ” de elemente din mulțimea I , căutarea într-un mod eficient tot este posibilă prin folosirea proprietății de “downward-closure” a suportului, numită și “anti-monotonie”. Aceasta garantează că pentru o mulțime de elemente frecvente, toate submulțimile acestea sunt de asemenea tot mulțimi de elemente frecvente. Această regulă aduce cu sine și reciproca sa, și anume : pentru o mulțime de elemente care nu sunt frecvente, nicio o mulțime mai mare în care ea este inclusă nu este de asemenea de elemente frecvente. Exploatând această proprietate, au fost creați o serie de algoritmi precum Apriori sau Eclat care pot să găsească toate mulțimile de elemente frecvente.

Pseudocod-ul algoritmului Apriori:

Procedure **Apriori**(T , minSupport){ // T este baza de date și minSupport este suportul minim

```

    L1 = {frequent items};
    for(k=2; Lk-1 != ∅; k++){
        Ck = mulțime candidată, generată din mulțimea Lk-1
        foreach tranzație t din baza de date do {
            Lk = mulțime candidată din Ck cu minSupport
        } // end for each
    } // end for
    return UkLk
}
```

Așa cum se întâmplă de obicei în căutarea și generarea de reguli de asociere, considerând un set de mulțimi de elemente (de exemplu mulțimi de tranzacții), un algoritm încearcă să găsească acele submulțimi care sunt comune în cel puțin “C” dintre acele mulțimi de elemente, unde C este un număr întreg inițial stabilit. Apriori utilizează o abordare bottom-up în care submulțimile frecvente sunt extinse cu câte un element la un moment dat, acest pas numindu-se “candidate generation”, după care se testează grupuri candidate față de datele inițiale. Algoritmul se încheie atunci când nicio extensie nu mai este găsită.

Apriori folosește căutarea în lățime și o structură de tip arbore pentru a număra mulțimile de elemente candidate într-un mod eficient. Acesta generează mulțimi candidate de lungime “k” pornind de la mulțimi de dimensiune “k-1”. Apoi, se elimină acei candidați care au un sub pattern infrecvent. Conform lemei de închidere (downward closure) , mulțimea candidată conține toate mulțimile de elemente frecvente de dimensiune “k”. După aceasta, se scanează baza de date care conține tranzacțiile pentru a determina care dintre mulțimile candidate sunt cu adevărat mulțimi de elemente frecvente.

Capitolul 5. Proiectare de Detaliu si Implementare

Acest capitol cuprinde detalii și informații referitoare la implementarea propriu-zisă a aplicației descrise în cadrul acestei lucrări. De asemenea, capitolul are ca obiectiv clarificarea conceptelor și descrierea tehnologiilor utilizate, prezentând funcționalitățile pe care aplicația le oferă. În prima parte se va descrie vederea generală asupra sistemului și modul de proiectare al acestuia, iar apoi se vor prezenta mai în detaliu modulele care îl alcătuiesc împreună cu operațiile implementate în cadrul acestora.

5.1. Implementarea arhitecturii sistemului

Implementarea acestei aplicații a fost realizată în limbajul de programare Java, respectând principiile de baza ale programării orientate pe obiecte, iar mediul de dezvoltare folosit a fost în mare parte Netbeans IDE 8. Sistemul a fost construit într-o manieră bottom-up, implementând mai întâi modulele mai mici iar apoi asamblându-le în scopul alcătuirii arhitecturii generale.

5.1.1. Vederea generală

Pentru început, va fi prezentată o vedere de ansamblu asupra aplicației. Aceasta a fost organizată în mai multe părți principale, atât pentru a respecta pattern-ul arhitectural ales Model-View-Controller cât și pentru a separa funcționalitățile aplicației unele de altele, în funcție de etapele în care se utilizează fiecare.

Astfel, putem distinge între trei mari părți care alcătuiesc aplicația :

- AbbreviationEditing
- ValidationRules
- ImportData

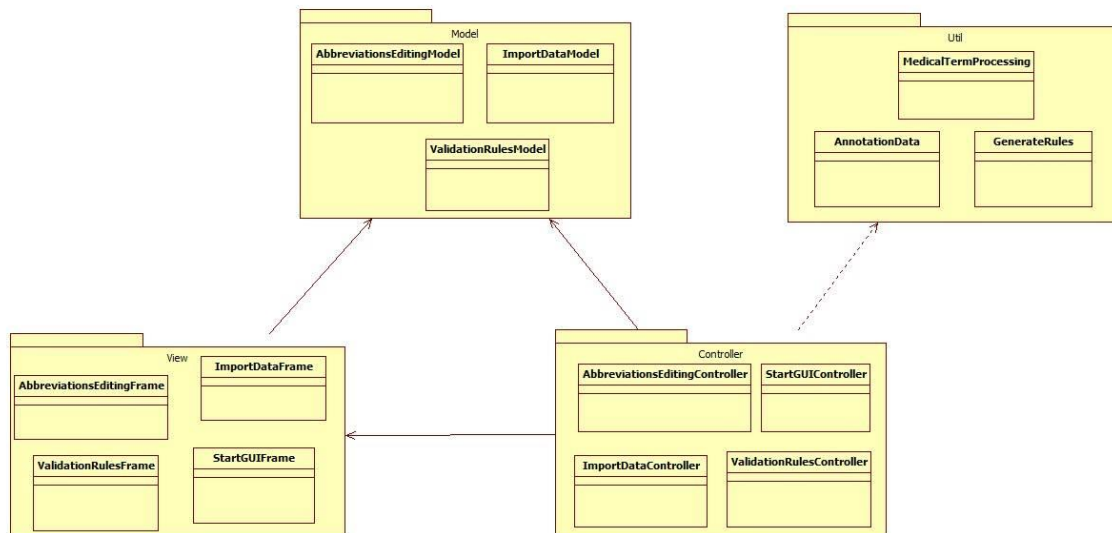


Figura 5.1 – arhitectura sistemului

Prima dintre acestea, așa cum am specificat și în cadrul capitolului de fundamente teoretice, va conține implementarea operațiilor care se realizează asupra dicționarului de abrevieri. Cea de-a doua parte se va ocupa de implementarea operațiilor necesare învățării de reguli, pornind de la studierea setului de date de învățare și până la manipularea regulilor generate. Nu în ultimul rând, a treia parte a sistemului este responsabilă atât cu înlocuirea abrevierilor în cadrul observațiilor medicale ce trebuiesc procesate, cât și cu generarea de adnotări pentru acestea.

De asemenea, pe lângă aceste părți principale, aplicația mai conține și un pachet de clase care au funcționalități generale, utile tuturor celor trei părți prezentate mai sus. Dacă aruncăm o privire mai amănunțită asupra design-ului, vom observa că fiecare dintre aceste părți prezentate este structurată în așa fel încât să separe logica de partea de vizualizare și de partea de control, însă aceste detalii vor fi prezentate mai pe larg în subcapitolul următor.

5.1.2. Implementarea pattern-ului Model-View-Controller

Acest pattern arhitectural, după cum am prezentat în capitolul 4, are rolul de a separa logica internă a unei aplicații de partea sa de prezentare. Scopul acestei separări este faptul că în proiectarea software, se dorește ca o componentă să fie capabilă în general să facă un singur lucru. Cu cât acea componentă știe să realizeze mai multe lucruri, cu atât codul devine greu de înțeles, de menținut și totul se complică. Prin folosirea pattern-ului MVC, partea de afișare și partea de date sunt menținute separat, astfel încât fiecare se poate schimba fără a o afecta pe cealaltă.

În cadrul acestei aplicații, prin utilizarea MVC se structurează fiecare modul component în trei părți, fiecare dintre acestea având câte un view, un model și un controller. View-urile mențin referințe către modelele lor corespunzătoare, iar controllerele mențin referințe atât către view-uri cât și către modele. De asemenea, pentru fațada aplicației se utilizează un view separat, cu un controller corespunzător, din cadrul căruia vor putea fi deschise celelalte trei module.

Așadar, prima etapă a fost crearea acestui view principal și a controllerului său. Având în vedere că această parte nu conține date sau logică suplimentară, ci doar legături către restul aplicației, nu avem nevoie de un model dedicat. La rularea aplicației se creează acest view numit StartGUIFrame, care este atașat unui obiect de tip StartGUIController. Frame-ul acesta principal conține 4 butoane cu ajutorul cărora se alege operațiunea dorită – continuarea înspre unul dintre module sau ieșirea din aplicație.

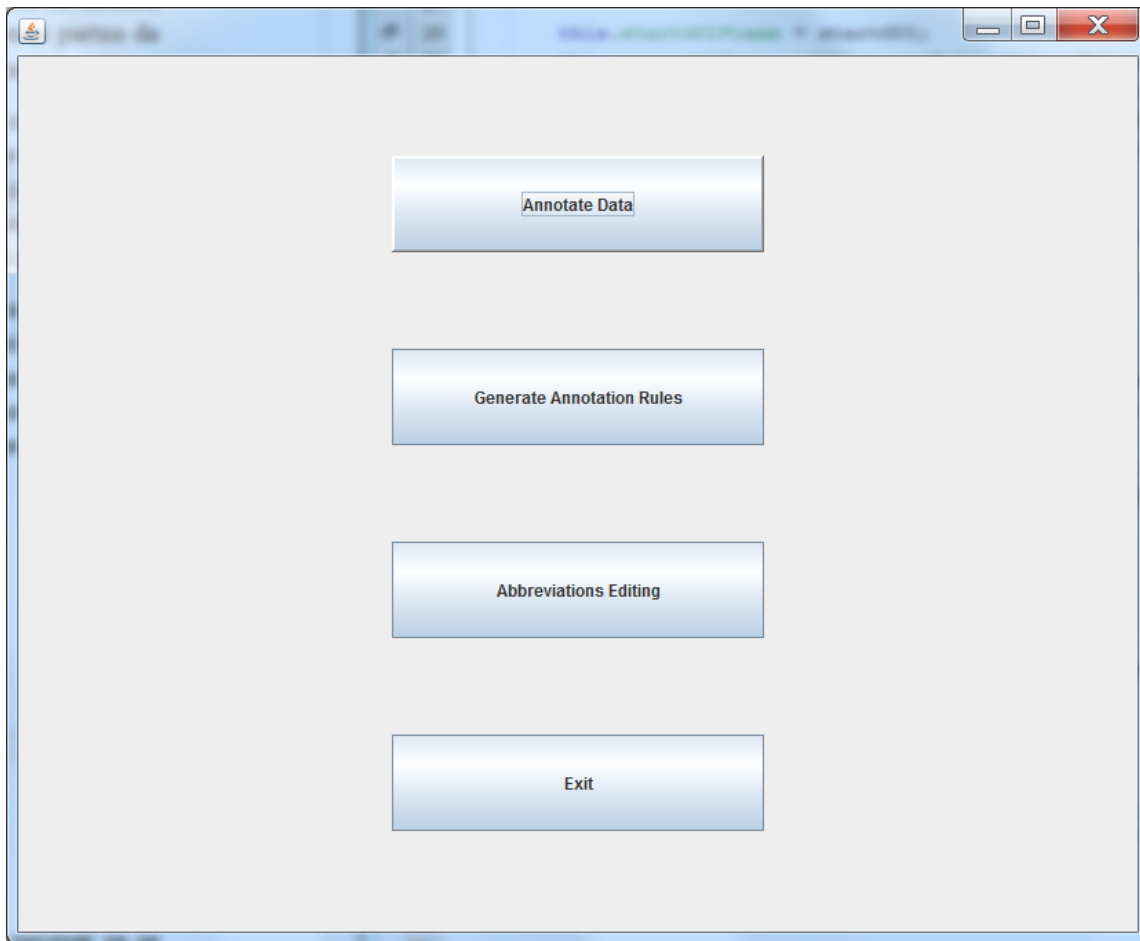


Figura 5.2 Frame-ul principal al aplicației

La alegerea unuia dintre butoanele care ne deschid celelalte module, este necesar să se creeze obiectele specifice acelei părți. Astfel, fiecare comandă va crea în spate modelul corespunzător, un view la care se va atașa acest model și apoi un controller la care se atașează atât view-ul cât și modelul. Un astfel de exemplu este prezentat mai jos:

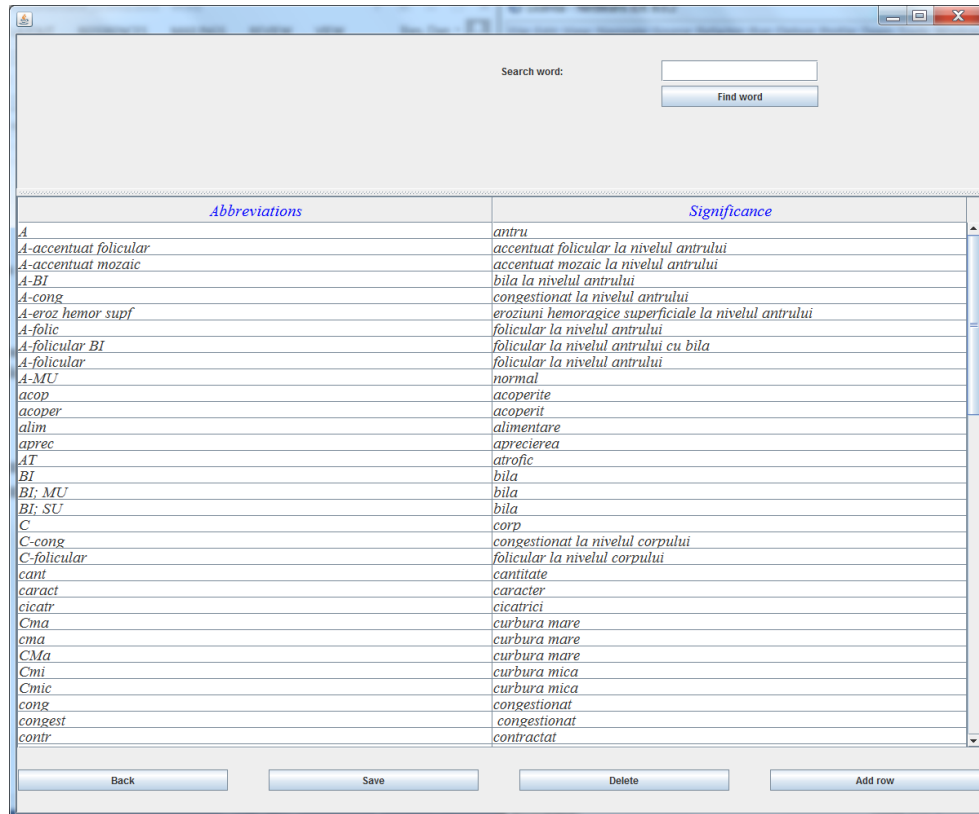
```
AbbreviationsEditingModel abrevModel = new AbbreviationsEditingModel();
AbbreviationsEditingFrame editAbrevieri =
    new AbbreviationsEditingFrame(abrevModel);
AbbreviationsEditingController abbrevEditController =
    new AbbreviationsEditingController(editAbrevieri, abrevModel);
```

În continuare urmează descrierea implementării și funcționalităților celor trei module principale ale aplicației.

5.1.3. Implementarea modului de abrevieri

Primul modul prezentat este cel de manipulare a abrevierilor. În cadrul acestuia se vor putea efectua operații asupra abrevierilor și semnificațiilor deja memorate, precum se vor putea și adăuga înregistrări noi. Interfața este una simplă și intuitivă, care oferă acces direct, prin butoane, către toate operațiile care se pot realiza.

Partea de view se rezumă la definirea, crearea și poziționarea elementelor din interfața grafică, precum și la definirea de metode de a adăuga listeneri pentru comenzile disponibile. Partea de controller se ocupă cu implementarea acestor listeneri, și cu operațiile mai simple care se pot efectua din acest modul, iar modelul are rolul de a păstra datele și de a realiza operațiile mai complexe.



Abbreviations	Significance
A	antru
A-accentuat folicular	accentuat folicular la nivelul antrului
A-accentuat mozaic	accentuat mozaic la nivelul antrului
A-BI	bila la nivelul antrului
A-cong	congestionat la nivelul antrului
A-eroz hemor supf	eroziuni hemoragice superficiale la nivelul antrului
A-folic	folicular la nivelul antrului
A-folicular BI	folicular la nivelul antrului cu bila
A-folicular	folicular la nivelul antrului
A-MU	normal
acop	acoperite
acoper	acoperit
alim	alimentare
aprec	aprecierea
AT	atrofic
BI	bila
BI; MU	bila
BI; SU	bila
C	corp
C-cong	congestionat la nivelul corpului
C-folicular	folicular la nivelul corpului
cant	cantitate
caract	caracter
cicatr	cicatrici
Cma	curbura mare
ema	curbura mare
CMa	curbura mare
Cmi	curbura mica
Cmic	curbura mica
cong	congestionat
congest	congestionat
contr	contractat

Figura 5.3 Interfața grafică a modulului de abrevieri

La deschiderea interfeței, se apelează automat operația de citire a abrevierilor existente și a semnificațiilor acestora din fișier, și afișarea lor în cadrul tabelului central, în ordine alfabetică. Această operație este implementată în cadrul modelului, și se efectuează utilizând obiectele pachetele java.nio și java.io. Înregistrările citite din fișier se salvează într-o listă, sunt sortate iar apoi afișate în tabel.

Operațiile mai simple precum adăugarea unei noi abrevieri sau ștergerea uneia existente sunt realizate direct în cadrul controllerului, întrucât nu necesită logică complexă. De asemenea, utilizatorul mai are posibilitatea de a sorta abrevierile alfabetic, în ordine crescătoare sau descrescătoare. Implementarea sortării s-a făcut cu ajutorul unui obiect de tip TableRowSorter căruia i s-a atașat tableModel-ul. Pentru această operație de sortare s-a creat și o nouă clasă care să implementeze interfața Comparator, cu scopul de a defini modul în care două obiecte sunt comparate.

Odată efectuate modificările dorite, utilizatorul are posibilitatea de a salva noul set de abrevieri în fișier, iar această operație este implementată în cadrul modelului. Practic, aceasta este inversul citirii, în sensul că se parcurge lista din cadrul tabelului și se scrie linie cu linie în fișier. Formatul fișierului va fi detaliat într-o secțiune viitoare care

se va concentra asupra seturilor de date. Nu în ultimul rând, interfața va permite căutarea unei anumite abrevieri în cadrul dicționarului pentru o mai bună accesibilitate. Această operație se realizează prin parcurgerea datelor din partea stânga a tabelului și verificarea potrivirii cu textul scris în căsuța de căutare.

5.1.4. Implementarea modului de generare de reguli

Cel de-al doilea modul prezentat este cel responsabil cu învățarea și generarea de reguli. Funcționalitățile acestuia se pot împărți în două etape, prima dintre ele fiind alegerea datelor dorite din setul de învățare și utilizarea acestora la generarea de reguli, iar cea de-a doua etapă fiind manipularea acestor reguli și salvarea lor, în scopul folosirii ulterioare. Interfața care oferă acces către aceste operații este asemănătoare cu a modului precedent, în sensul că avem un tabel în centrul acesteia care va fi folosit la afișarea regulilor generate, iar în jurul tabelului vor fi situate elementele care permit accesul la operații, cum ar fi butoane sau jComboBox-uri.

Rules	Support	Select
S-Lum VGI <- mucoasa	31.5789	<input type="checkbox"/>
S-Lum VGI <- partial stomach	57.8947	<input type="checkbox"/>
S-Lum VGI <- partial volvulat	73.6842	<input type="checkbox"/>
S-Lum VGI <- partial volvulat stomach	57.8947	<input type="checkbox"/>
S-Lum VGI <- volvulat	73.6842	<input type="checkbox"/>
S-Lum VGI <- volvulat stomach	57.8947	<input type="checkbox"/>

Figura 5.4 Interfața grafică a modului de generare de reguli

Inițial, tabelul este vid întrucât generarea regulilor are loc abia după ce utilizatorul selectează valorile dorite. Singură operație care se realizează automat este popularea ComboBox-ului din stânga sus cu prescurtările care apar în setul de învățare. După selectarea valorii dorite, se vor parcurge datele setului de învățare pentru respectiva valoare și se vor citi toate adnotările corespunzătoare. Această prelucrare are loc cu

ajutorul modelului, care implementează setul de operații mai complexe din cadrul acestui modul, și are loc în felul următor:

În prima fază se citește prima linie din setul de date pentru realizarea matching-ului între valoarea aleasă și coloana corespunzătoare. Apoi, se ia fiecare linie în parte și se citește atât valoarea de pe prima coloană, care reprezintă observația medicală, cât și valoarea din coloana determinată anterior, care semnifică adnotarea corespunzătoare. În acest fel, vom obține perechi de tip observație-adnotare, pe care le vom păstra în 2 liste construite în paralel. Odată terminat acest proces, am decis că aceste valori citite să fie extrase separat într-un fișier care să conțină doar observațiile împreună cu acea coloană de adnotări, fiind astfel separate doar valorile de interes pentru acest caz.

După această separare, trebuie să se populeze cel de-al doilea ComboBox cu valorile tuturor adnotărilor întâlnite la pasul anterior, așadar va trebui să se facă o extragere a acestora din fișierul nou generat. Pentru asta am implementat o nouă metodă în cadrul modelului căreia îi transmitem ca parametru numele coloanei din urma căreia să se facă extragerea, întrucât acesta va fi specificat în numele fișierului salvat anterior. Așadar, acest fișier se va parcurge linie cu linie, iar în momentul în care întâlnim o abreviere pe care nu am mai întâlnit-o deja, o adăugăm într-o listă. În urma acestei operații vom obține în final reuniunea tuturor abrevierilor, care o putem utiliza apoi pentru popularea celui de-al doilea ComboBox.

În momentul în care avem deja toate opțiunile disponibile, utilizatorul are posibilitatea de a porni procesul de generare de reguli. Acest proces va folosi implementarea algoritmului Apriori pentru a procesa datele setului de învățare și pentru a găsi elementele frecvente din cadrul acestuia. Trebuie să menționăm faptul că la prima aplicare, înainte de a rula algoritmul propriu-zis asupra datelor, aplicația va procesa fișierul generat la pasul anterior, de selectarea a valorilor dorite, care conține informații despre o anumită coloană din setul de învățare împreună cu toate valorile posibile pentru aceasta (de exemplu, conține și S-Lum_VGI + observația, și S-Lum_RDG + obs, și S-Lum_DIL + obs). Prin această procesare ne referim la acțiunea de a selecta din cadrul acestui fișier doar acele înregistrări care corespund cu adnotarea aleasă de utilizator (de exemplu, de data asta se aleg doar liniile care conțin S-Lum_DIL + observația). După ce se realizează selecția prin parcurgerea fișierului și memorarea înregistrărilor corespunzătoare într-o listă, acestea vor fi scrise într-un nou fișier care va reprezenta setul de date pe care se aplică efectiv algoritmul Apriori. Apoi, la următoarele aplicări pentru aceleași valori, nu va mai fi nevoie de această procesare de fișier în prealabil, întrucât vom avea deja selectate înregistrările dorite. Această diferențiere între prima rulare, unde e nevoie de acel pas suplimentar, și restul rulărilor e stabilită cu ajutorul unui flag care își schimbă valoarea inițială după ce algoritmul este aplicat prima dată, și revine la această valoare inițială abia atunci când utilizatorul selectează alte valori pentru care să se facă extragerea de reguli.

Pentru execuția algoritmului Apriori este necesar să îi setăm o serie de parametri, iar cei modificabili de către utilizatori sunt suportul și încrederea (confidence). Rolul acestora a fost descris în capitolul de fundamente teoretice, așadar acum menționăm doar că aceștia se citesc din interfața grafică după introducerea lor în căsuțele dedicate, și sunt folosiți la execuția algoritmului. Această execuție este pornită în cadrul clasei GenerateRules, special creată în acest sens, având două metode dedicate : una pentru prima rulare după ce utilizatorul a ales valori noi (cu alte cuvinte – când flag-ul are

valoarea inițială) iar cealaltă pentru rulările ulterioare ținând cont de aceleași valori. Lansarea propriu-zisă în execuție a algoritmului și modul în care sunt generate rezultatele vor fi descrise mai în detaliu în secțiunea 5.3.

După prima rulare, vom obține un set de reguli frecvente care se vor afișa în tabelul central prin citirea din fișierul de ieșire creat anterior. Din cadrul acestora, utilizatorul are posibilitatea să le aleagă pe acelea pe care le consideră mai relevante și să le salveze. Salvarea are loc într-un fișier special dedicat coloanei alese inițial, care să conțină regulile relevante pentru respectiva coloană, ce vor urma să fie utilizate la etichetare. Procesul de salvare are loc prin selectarea regulilor bifate și adăugarea acestora la fișierul corespunzător (sau crearea acestuia în cazul în care nu există). Trebuie să mai menționăm că regulile alese, după memorare, vor fi șterse atât din setul de date pe care se aplică Apriori cât și din tabel, întrucât se consideră că s-au validat deja, iar la următoarele aplicări ale algoritmului să nu se genereze din nou.

După memorarea regulilor dorite și ștergerea lor din datele de învățare, utilizatorul poate genera din nou un set de reguli din datele rămase, și să repete procesul de memorare. Această buclă se poate repeta de mai multe ori, până când utilizatorul consideră că a validat toate regulile relevante. Când acest proces se încheie, putem considera că etapa de învățare s-a terminat, pentru acele valori selectate, și se poate merge mai departe la partea de adnotare a observațiilor.

5.1.5. Implementarea modului de adnotare

Al treilea modul al sistemului este cel responsabil cu procesarea observațiilor care trebuie adnotate. Întrucât aceste observații sunt scrise în limbaj natural folosind o mulțime de prescurtări, procesarea implică înainte de toate o parcurgere a acestora și o înlocuire a cuvintelor prescurtate cu semnificația lor completă, pe baza dicționarului prezentat în cadrul modului de abrevieri. După ce această operațiune are loc, urmează aplicarea regulilor obținute în pasul anterior peste aceste observații, cu scopul de a le eticheta. Interfața grafică a modului prezintă, la fel ca și în cazurile anterioare, un tabel central și comenzi disponibile în jurul acestuia.

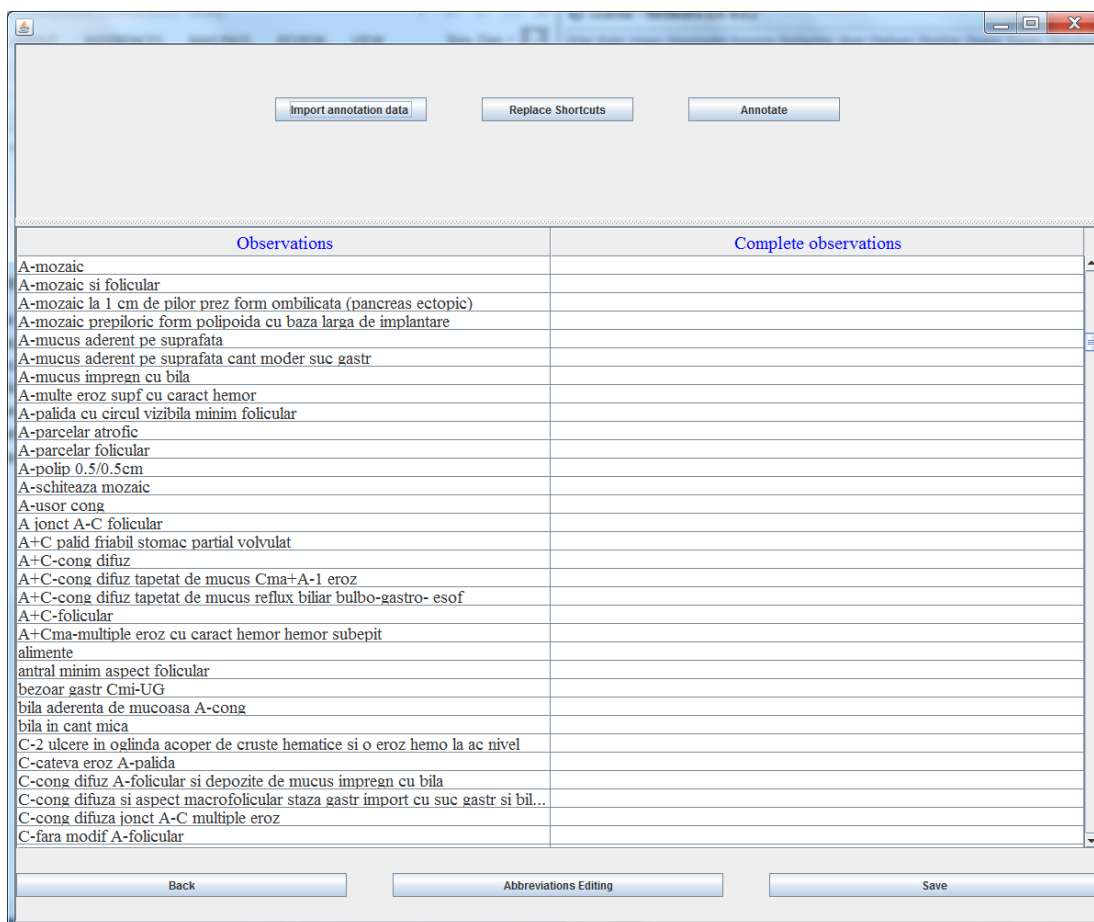


Figura 5.5 Interfata modulului de adnotare

Primul pas, înainte de toate, în lucrul cu acest modul îl reprezintă încărcarea fișierului care conține înregistrările ce vor fi procesate, acesta realizându-se prin intermediul unui FileChooser. Odată selectat, fișierul respectiv va fi citit și afișat în tabelul central.

Întrucât dicționarul de abrevieri se presupune că a fost deja creat și populat cu ajutorul primului modul descris, înlocuirea abrevierilor are loc într-un singur pas. Totuși, algoritmul care stă în spatele înlocuirii nu este la fel de simplu. Aceste abrevieri pot apărea într-o multitudine de forme, de la cele simple (A, BI, etc) până la cele compuse (A-folicular). De asemenea, se pot și conține unele în altele (RDG, RDGE) așadar a fost nevoie de o procesare atentă a acestora.

Ideea principală a fost de a lua fiecare abreviere în parte și de a înlocui apariția ei peste tot în fișier unde aceasta e prezentă. Pentru a evita problemele care ar putea să apară atunci când avem o abreviere simplă care este prezentă și într-o construcție compusă (A și A-folicular) și a căror semnificație nu este identică, am ales că prima dată să se detecteze abrevierile care sunt compuse și să se înlocuiască acelea întâi. Astfel, se elimină cazurile în care abrevierile compuse ar fi înlocuite defapt în doi sau mai mulți pași.

De asemenea, pentru a reține la fiecare pas înlocuirile făcute la pasul anterior, lista de observații va fi modificată dinamic, însă avem nevoie și de varianta sa inițială pentru a păstra prima coloană din tabel intactă. Lista modificată dinamic va fi parcursă cu

un iterator, iar linia curentă din fișierul de abrevieri pentru care se face înlocuirea va fi împărțită în două : abreviere și semnificație. Logica de înlocuire este următoarea : în cazul în care observația nu conține abrevierea, se păstrează în formă în care e și se trece la observația următoare. În cazul în care o conține, începe logica de înlocuire.

Având în vedere multitudinea de forme sub care pot să apară și faptul că se pot conține unele în altele, cel mai eficient mod de a detecta prezența abrevierilor este folosind structuri Regex – Pattern, Matcher, întrucât cu ajutorul acestora putem căuta cuvinte într-o varietate de forme. După potrivirea abrevierii peste apariția ei în observație, aceasta este înlocuită cu semnificația și memorată astfel încât modificările să fie disponibile la pasul următor. În final, după încheierea înlocuirilor, pe a doua coloană a tabelului vor apărea observațiile în formă completă. Nu în ultimul rând, trebuie să menționăm faptul că din interfața acestui modul avem o scurtătură către dicționarul de adnotări, pentru cazurile în care se dorește efectuarea unor modificări rapide.

După încheierea acestei etape, urmează partea de adnotare a textului. Aceasta va avea loc pentru acele coloane care sunt selectate de utilizator, și se face în mod automat la adăugarea respectivelor coloane. La alegerea unei coloane, se verifică numele acesteia iar apoi se caută fișierul corespunzător cu reguli, după care să se facă etichetarea. În cazul în care fișierul nu e găsit, se afișează un mesaj de eroare, iar dacă e găsit atunci se face din nou o potrivire între partea dreapta a regulilor și observații. Unde se întâlnește o astfel de potrivire, se completează cu partea stânga a regulii în coloana de adnotări. La citirea fișierului cu reguli, acestea se memorează într-un Map, în perechi de tip : partea dreapta – partea stânga a regulii, pentru un acces mai ușor la ele. În urma adnotării coloanelor dorite, utilizatorul are posibilitatea de a salva datele obținute într-un nou fișier.

5.2. Modulul de prelucrare a datelor

În subcapitolul anterior am vorbit despre modul în care a fost implementată aplicația și despre cum lucrează aceasta cu setul de date. Nu am intrat însă în detaliile de organizare ale acestor date asupra cărora operează aplicația, ci am descris doar modul în care acestea sunt prezentate utilizatorului prin intermediul interfeței grafice.

În cadrul acestui subcapitol se va prezenta mai pe larg setul de date, împreună cu modalitățile prin care acestea sunt prelucrate, pornind de la primele date pe care le utilizează aplicația, trecând prin cele create în pașii intermediari și finalizând cu prezentarea datelor de ieșire. Pe lângă structura lor, se va discuta și despre tehnicile de interpretare folosite la prelucrarea acestora și despre semnificațiile pe care le au structurile la fiecare pas.

Ca și aspecte generale, trebuie să menționăm faptul că datele sunt în general stocate în fișiere .csv, cu valori separate prin virgulă, ceea ce le face mai ușor de citit și extras. Pe lângă acest tip de date, mai avem câteva tipuri proprii cu extensiile .in sau .out, acestea fiind folosite în principal de partea Apriori din cadrul aplicației, dar nu numai. În continuare vom începe cu descrierea datelor primului modul și modul de utilizare al acestora.

5.2.1. Prelucrarea datelor de învățare

Datele de învățare au unul dintre cele mai importante roluri în buna funcționare a aplicației. Acest lucru se datorează faptului că, inițial, sistemul nu pornește cu niște cunoștințe predefinite pe baza cărora să știe să facă adnotarea. În schimb, el are nevoie de acest set inițial de date de pe urma căruia să învețe cum ar trebui să realizeze procesul de adnotare, prin studierea a zeci și sute de exemple deja etichetate. Așadar, exact asta trebuie să conțină setul de date de învățare : observații medicale de tipul celor pe care sistemul le va analiza mai târziu, iar acestea să fie deja adnotate pentru a se putea deduce informații de pe urma lor.

Pentru început, prezentăm structura acestor date de învățare, ele fiind salvate în cadrul unor fișiere cu extensia .csv. Asta înseamnă că informația este organizată pe linii iar grupurile separate de date de pe aceeași linie sunt separate prin virgulă. Modul de reprezentare al datelor în aceste fișiere este sub forma :

Observatie – eticheta1 – eticheta2 – eticheta3 ...

Prima linie a fișierului de învățare va arăta în felul următor:

Stomac,S-Lum,S-Cont,S-MucANT,S-MucJAC,S-MucCORP,SlezANT,S-LezJAC,S-LezCORP,Pilor

Practic, prima linie din fișier joacă rolul unui antet de tabel, în sensul că specifică semnificația fiecărui element de pe liniile următoare în funcție de poziția aceluia element în cadrul liniei. De exemplu, dacă pe prima linie avem “S-Lum” ca fiind al doilea element, asta înseamnă că în restul fișierului, pe fiecare linie cel de-al doilea element reprezintă eticheta corespunzătoare pentru “S-Lum”, pentru observația medicală de pe acea linie. Dacă se respectă acest format, aplicația poate apoi implementa un mod general de parsare al acestor date de învățare, în funcție de alegerile utilizatorului. Preluarea de informații din aceste fișiere de învățare are loc în două etape.

Astfel, prima etapă este reprezentată de descoperirea valorii pentru care se face căutarea, adică găsirea poziției pe care se află acea valoare în cadrul antetului. După aflarea acestei informații, a doua etapă constă în păstrarea indexului poziției respective iar apoi la citirea celorlalte linii, se ia de pe fiecare observația medicală împreună cu eticheta corespunzătoare indexului memorat anterior. În acest fel, pentru valoarea aleasă de utilizator vom obține o serie de perechi observație – etichetă, de pe urma cărora se poate realiza învățarea și generarea de reguli de adnotare.

Pentru a ne putea folosi de aceste perechi extrase într-o etapă următoare, este necesară memorarea acestora undeva. Ele vor fi utilizate în procesul de creare de reguli, iar acest proces se realizează prin aplicarea algoritmului Apriori peste acestea. Pentru a le putea transmite către algoritm, nu este suficientă păstrarea lor într-o structură de date din limbajul Java, întrucât Apriori are nevoie de date de intrare în format text, memorate într-un fișier. În plus, la selecția ulterioară a regulilor se va dori manipularea și acestor perechi, de exemplu ștergerea acelor care se potrivesc cu o regulă selectată de utilizator. Având aceste constrângeri, am decis că perechile să fie memorate într-un fișier separat, acesta având numele : Data.numeColoana.in , și conținând perechile inversate, adică : eticheta – observație.

Structura organizată a acestor date de intrare ne permite să le parsăm într-un mod relativ simplu, iar acum că am descris această etapă, putem trece mai departe la

următoarele seturi de date, și anume la cele de pe urma cărora se realizează înlocuirea abrevierilor.

5.2.2. Algoritmul de înlocuire al abrevierilor

În descrierea implementării aplicației, cu precădere în cadrul modulului de abrevieri, am adus aminte de multe ori de așa-zisul dicționar de abrevieri. Acesta este o structură de date care conține prescurtări apărute în observații medicale, împreună cu semnificațiile lor. Scopul lui este de a fi utilizat într-o etapă care precede adnotarea observațiilor, pentru a înlocui în cadrul acestora cuvintele prescurtate cu semnificațiile acestora.

Dicționarul de abrevieri este păstrat într-un fișier cu extensia .csv, acesta conținând informațiile într-un mod asemănător cu fișierele descrise în secțiunea anterioară. De această dată însă, fiecare linie conține exact două elemente: o prescurtare pe prima poziție și definiția acesteia pe a doua poziție.

La citirea acestor abrevieri, ele sunt memorate temporar într-o structură internă de date, și anume o listă, fiecare element al acesteia conținând un singur rând din fișier. Când se dorește procesarea acestei liste și separarea între prescurtare și semnificație, se utilizează funcția “split” din java pentru a împărți fiecare element al listei în cele două părți.

Această operație e necesară pentru a avea valorile separate, întrucât abrevierea trebuie folosită la căutarea printre observații în scopul înlocuirii acesteia. Înainte de a trece la procesul propriu-zis de înlocuire al abrevierilor, mai amintim faptul că acest dicționar în format .csv poate fi modificat de către utilizator, realizând operații de ștergere, editare sau creare. După încheierea și salvarea actualizărilor, utilizatorul poate trece la partea de înlocuire a prescurtărilor.

Algoritmul care realizează această înlocuire trebuie să țină cont de mai mulți factori. În primul rând, este posibil ca într-o înregistrare să apară mai multe prescurtări, așadar nu este suficientă o singură înlocuire per rând. Apoi, o a doua problemă este cea a cuvintelor compuse, întrucât pot să apară și astfel de structuri, și acestea trebuie interpretate ca un întreg, nu ca și două părți separate prin liniuță sau orice alt delimitator. De asemenea, este posibil să avem și prescurtări imbricate, cum ar fi : RDG și RDGE. În acest caz, o problemă care ar putea să apară ar fi faptul că la întâlnirea formei mai lungi (RDGE), această să fie înlocuită cu forma cea scurtă și să rezulte un cuvânt inconsistent (reflux duodeno-gastricE). Rezolvările acestor posibile probleme vor fi prezentate în cele ce urmează.

Informațiile necesare vor fi întâi citite din fișierele corespunzătoare iar apoi vor fi memorate în structuri de date ale limbajului Java – liste, pentru o procesare mai rapidă și mai lipsită de riscuri comparativ cu posibilitatea de a efectua modificări direct pe setul de date, adică pe fișierul de observații. Ca mod de funcționare, algoritmul de înlocuire al abrevierilor va lua fiecare înregistrare în parte din dicționar și va verifică prezența prescurtării din acea înregistrare în fiecare observație din setul pe care se aplică. În cazurile în care se realizează o potrivire, acea prescurtare e înlocuită cu semnificația ei. Această operație de înlocuire se realizează asupra structurilor în care s-au memorat

informațiile și nu în cadrul fișierului. După încheierea procesării și obținerea datelor în formă finală, abia atunci acestea vor putea fi scrise în fișierul cu rezultate.

Pentru rezolvarea primei probleme descrise mai sus, am decis că lista de observații să fie modificată în mod dinamic, dacă e cazul, în timpul parcurgerii și verificării prezenței unei anumite adnotări. În acest scop am folosit un Iterator pentru parcurgea acestei liste, întrucât printr-o astfel de interfață avem certitudinea că modificările noastre nu alterează structura în sine. Atunci când o abreviere este găsită pe un anumit rând, se înlocuiește cu semnificația ei și se actualizează lista de observații. În acest fel, la următorul pas când parcurgem observațiile verificând prezența unei alte prescurtări, avem deja modificările din urma abrevierii precedente. Acest lucru creează o reacție în lanț în așa fel încât în final, se înlocuiesc oricât de multe abrevieri pe un rând. Totuși, trebuie să avem grijă la faptul că vom avea nevoie de formă inițială a observațiilor și după încheierea algoritmului de înlocuire. Așadar, având în vedere cele spuse mai sus cu privire la modificarea dinamică a listei de observații, mai avem nevoie și de o structură de date care să păstreze observațiile în forma inițială. Această structură va fi reprezentată de o listă de obiecte de tip String.

Cea de-a doua problemă apare în cazurile acelor prescurtări care sunt prezente atât individual cât și într-o formă compusă. Întrucât majoritatea au o semnificație diferită în formă compusă decât dacă s-ar înlocui cuvintele mot-a-mot (de exemplu, A = antru, BI = bilă iar A-BI = bilă la nivelul antrului, și nu: antru-bilă), trebuie să avem grijă că nici înlocuirea să nu se facă treptat, adică pe bucăți. În acest scop am găsit două soluții, una fiind cea de ordonare a abrevierilor în dicționar în așa fel încât acelea compuse să fie verificate primele. În acest fel se vor înlocui întâi prescurtările compuse, și se va elimina posibilitatea de a le înlocui pe bucăți. Totuși, această soluție nu a fost aleasă, în special din cauza faptului că într-un dicționar, termenii sunt de obicei aranjați alfabetic, fără reguli suplimentare precum ar fi fost această , adică acele abrevieri care sunt cuvinte compuse să fie primele. Abrevierile au fost lăsate în ordine alfabetică, iar soluția constă în următorul artificiu : la parcurgerea listei de abrevieri, se detectează mai întâi care dintre acestea sunt cuvinte compuse, indiferent de poziția lor în setul de date, și se rulează algoritmul de înlocuire doar pentru acestea, în prima fază. Apoi, după ce prescurtările de acest tip au fost înlocuite, se parcurge dicționarul în mod obișnuit și se înlocuiesc celelalte înregistrări, care de data aceasta sunt abrevieri simple. Întrucât la pasul anterior au fost înlocuite deja abrevierile compuse (de tip A-BI sau A-folcular), în acest al doilea pas nu vom mai avea prescurtări într-o asemenea formă compusă, ci doar simple și de sine stătătoare. Așadar, acest artificiu prezentat rezolvă problema numărul doi pe care am menționat-o mai sus.

Rămâne așadar de discutat a treia problemă dintre cele prezentate inițial, și anume problema abrevierilor care sunt conținute una în alta, precum RDG și RDGE. În mod asemănător cu a doua problemă, fără o optimizare necesară algoritmul ar avea șansa să detecteze prescurtarea RDGE atunci când caută printre observații după abrevierea RDG, și să facă o înlocuire necorespunzătoare. În acest caz însă, nu mai putem aplica soluția de mai sus. Atunci era posibilă detectarea cuvintelor compuse examinând forma acestora, însă de data aceasta este dificil să ne dăm seama dacă o abreviere conține, în structura ei sintactică, o altă abreviere mai scurtă. Pentru a ne asigura că abrevierile mai lungi se vor înlocui corespunzător, o posibilă soluție ar fi ordonarea acestora în cadrul dicționarului în așa fel încât cele mai lungi să apară primele, iar cele scurte mai la final. Asta ar asigura

înlocuirea corespunzătoare a lor, însă ne-ar genera din nou problema întâlnită mai sus, și anume faptul că abrevierile nu vor mai fi în ordine lexicografică. Așadar, soluția aleasă nu s-a bazat pe această idee, ci a constatat în optimizarea detecției abrevierilor în așa fel încât atunci când algoritmul detectează o posibilă potrivire, înlocuirea să aibă loc doar în cazul în care potrivirea este exactă, și nu parțială. Acest lucru a putut fi realizat prin expresii regulate, adică mai exact prin structurile Pattern și Matcher puse la dispoziție de limbajul Java. Pattern-ul este o reprezentare compilată a unei expresii regulate definite de către programator, iar matcher-ul reprezintă un tip de obiect creat de pe urma unui Pattern, care este folosit pentru a căuta în cadrul unui text aparițiile pattern-ului respectiv din care a fost creat. În cazul implementării de față, acestea vor fi folosite în felul următor : la parcurgerea abrevierilor, după stabilirea abrevierii curente pentru care se face căutarea în acest pas, se creează un pattern de pe urma acesteia. Acest pattern însă nu este creat doar după numele abrevierii, ci i se stabilește atât la început cât și la sfârșit un caracter special “b”, care să reprezinte margine de cuvânt. Apoi se creează un matcher de pe urma acestui pattern, iar căutarea în cadrul observațiilor a abrevierii se va face prin metoda “find” a obiectului de tip Matcher. Prin acest mod de căutare, ne vom asigura de faptul că eventuala abreviere găsită e cuvânt de sine stătător, așadar nu este doar o parte dintr-o abreviere sau un cuvânt mai mare. La realizarea unui matching, așa cum am menționat și mai sus, se va înlocui acea abreviere cu semnificația ei.

Aceste posibile probleme fiind tratate, algoritmul de înlocuire va funcționa în mod corespunzător. După procesarea întregului set de date, vom obține o reprezentare a acestuia cu abrevierile înlocuite, într-o listă din cadrul programului. Aceste date se vor păstra pentru moment doar în memoria de lucru, întrucât operația de înlocuire a abrevierilor se realizează de cele mai multe ori înainte de procesul de adnotare. Dacă totuși se dorește salvarea acestora înainte de adnotare, operația este permisă iar observațiile înlocuite se vor scrie într-un fișier de ieșire.

După realizarea operației de înlocuire a abrevierilor, utilizatorul poate trece la următoarea etapă. Aceasta va consta în generarea de adnotări specifice pentru observațiile medicale care au fost procesate în acest pas de înlocuire. Modul de realizare al adnotării va fi descris în cele ce urmează.

5.2.3. *Procesul de adnotare al datelor*

Acesta este unul dintre cele mai semnificative procese pe care le execută aplicația de la baza acestei lucrări. Practic, pentru acest pas s-au făcut toate pregătirile anterioare asupra datelor, și anume :

- Crearea dicționarului de abrevieri
- Obținerea unui modul de date deja adnotate, în scopul învățării de reguli
- Învățarea de reguli
- Înlocuirea abrevierilor care apar în setul de observații pe care aplicația trebuie să îl proceseze.

Așadar, fiind nevoie de toți acești pași premergători, putem spune că procedeul prin care se adnotează datele medicale reprezintă pasul final al fluxului aplicației.

Spre deosebire de procesele descrise anterior, acest proces de adnotare nu are datele de intrare salvate într-un fișier, cel puțin nu în mod direct. El se va aplica pe un set

de observații medicale care sunt inițial memorate într-un fișier .csv, însă nu pe forma din fișier a acestora, lucru datorat faptului că în fișier datele sunt salvate în forma cu prescurtări. Așadar, adnotarea se va face în urma procesului de înlocuire a abrevierilor, iar rezultatele obținute nu se mai memorează într-un fișier intermediar. În schimb, acestea sunt păstrate în memorie după ce înlocuirea abrevierilor se încheie și folosite mai apoi la generarea adnotărilor.

Pe lângă datele din memorie menționate mai sus, procesul de adnotare mai are nevoie și de reguli. Acestea trebuie să fie generate în prealabil, în urma prelucrării setului de date de învățare și aplicării algoritmului Apriori pe acestea, pentru a găsi elementele comune care duc de cele mai multe ori la o etichetă sau alta. În secțiunea 5.2.1 am specificat modul de obținere al informațiilor din setul de date de învățare și faptul că algoritmul Apriori va genera, pornind de la acestea, un nou fișier care conține reguli de adnotare, acesta fiind cel care trebuie citit pentru că procesul de adnotare să poată avea loc. Structura unei linii arată în felul următor:

Eticheta <- cuvinte cheie. (S-Lum_RDG <- duodeno gastric).

Acest fișier va fi parcurs iar conținutul său va fi salvat în memorie pentru utilizarea ulterioară. Judecând după modul în care sunt organizate regulile, am ales ca structura de date în care se vor memora să fie un HashMap. În Java, un Map în general reprezintă o structura cu înregistrări de tip cheie-valoare. În cazul nostru, întrucât o eticheta poate fi generată de mai multe grupuri de cuvinte cheie diferite, nu ar fi o soluție bună să alegem eticheta ca fiind cheia Map-ului, întrucât numărul de coliziuni ar fi prea mare. În schimb, vom alege ca grupurile de cuvinte cheie să aibă acest rol, iar etichetele să fie valorile. Această alegere este mai benefică și din cauza faptului că atunci când se realizează etichetarea, algoritmul verifică defapt observația medicală și pe baza cuvintelor caută eticheta corespunzătoare.

Așadar, avem într-o structura observațiile cu abrevierile deja înlocuite, și în altă structură avem regulile. Aceste reguli vor fi obținute din fișierul corespunzător valorii coloanei pe care o alege utilizatorul, vor fi împărțite după "<-" și apoi memorate în HashMap. După asta, algoritmul de generare de adnotări va parcurge observațiile și la fiecare înregistrare va caută în Map o posibilă etichetă corespunzătoare. În cazul descoperirii uneia, o completează în coloana corespunzătoare și în structura din spatele tabelului, iar altfel se va completa cu o valoare implicită.

După realizarea generării de adnotări, la alegerea opțiunii de salvare se vor parcurge observațiile în formă completă și etichetele generate pentru acestea, în scopul salvării lor într-un fișier de ieșire. Acesta va fi rezultatul final al aplicației, adică setul de observații adnotate.

5.2.4. Modul de integrare al algoritmului Apriori

În cadrul secțiunii 5.1 când s-a descris implementarea arhitecturii sistemului, am menționat în cadrul modulului de învățare că se utilizează o implementare a algoritmului Apriori care să analizeze setul de date de învățare și să găsească grupuri de cuvinte cheie frecvente care duc înspre aceleași adnotări. În această secțiune vom prezenta mai în detaliu modul de integrare al acestei implementări în aplicația noastră, motivul pentru care am ales varianta aceasta de implementare în detrimentul altora, cum ar fi cea din

biblioteca Weka, precum și alte operații adiționale care sunt necesare la utilizarea algoritmului.

Secțiunea de cod care se ocupă de apelarea Apriori a fost proiectată separat de modulele principale, fiind definită o clasă specială în acest sens. În cadrul acesteia am implementat o logică de apelare a executabilului “apriori.exe” prin intermediul clasei Runtime pusă la dispoziție de limbajul Java. Cu ajutorul acesteia, se lansează în execuție un proces care să poată realiza operații într-un mod similar cu cel în care utilizatorul ar efectua aceste informații dintr-o fereastră “command prompt”. Pe lângă apelul propriu-zis al executabilului, se mai execută o serie de comenzi care să proceseze informațiile din fișiere. Aceste comenzi selectează etichetele corespunzătoare coloanei alese de utilizator împreună cu observațiile, salvându-le într-un fișier intermediar. Apoi urmează rularea executabilului de Apriori pe acel fișier, împreună cu parametri corespunzători, ale căror valori au fost alese de utilizatori și transmise acestei clase. Metodele de apelare ale algoritmului nu returnează niciun fel de informație, ci doar execută comenzi ca și niște proceduri. În final, algoritmul va genera un fișier cu rezultatele aplicării sale, fișier care va conține mulțimi de cuvinte frecvente care apar în setul de date inițiale. Acestea sunt organizate ca niște reguli, iar în după generarea lor mai urmează un ultim pas în care acestea sunt selectate în funcție de valoarea aleasă inițial de utilizator, iar apoi sunt sortate pentru a fi organizate într-o formă mai lizibilă.

Ca și implementare a algoritmului Apriori, mai aveam o variantă a acestuia implementată direct în Java, prin biblioteca Weka. Totuși, am preferat să nu se utilizeze acea variantă întrucât aceea are nevoie de un fișier special de tip .arff, care să aibă o structură proprie specifică, iar maparea ei pe setul nostru de date era destul de dificilă. De asemenea, implementarea din Weka nu este nici la fel de performantă în ceea ce privește timpul de execuție.

Capitolul 6. Testare și Validare

În procesul de dezvoltare al acestei aplicații, cu toate că s-a acordat o atenție sporită la implementarea algoritmilor și operațiilor necesare, a fost nevoie și de o continuă testare a codului pentru a avea siguranța că nu rămân în urmă erori.

6.1. Testarea generală a codului

Pentru proiectarea și dezvoltarea aplicației am folosit un mediu de dezvoltare special dedicat limbajului Java, și anume Netbeans IDE. Cu ajutorul acestuia, am reușit să văd în timp real dacă apăreau greșeli de tipar sau dacă anumite părți ale codului nu erau corecte sau complete. În plus, Netbeans oferă și un modul de debug cu ajutorul căruia se poate testa pas cu pas execuția aplicației. Acest modul s-a dovedit a fi foarte util în cazurile în care aplicația nu funcționa în totalitate conform așteptărilor, întrucât oferă acces către valorile variabilelor de la un moment dat și către stiva de apeluri de metode.

În plus, pe lângă aplicația principală am mai implementat și câteva proiecte mai mici în care am testat anumite funcționalități specifice, de exemplu citirea dintr-un fișier sau scrierea în acesta, algoritmi de detectare de String-uri și așa mai departe.

6.2. Testare funcțională

Pentru a ști cu siguranță că aplicația funcționează corect și conform așteptărilor, am realizat o testare amănunțită a fiecărei părți componente și a rezultatelor pe care acestea le generează, comparându-le cu ceea ce se aștepta defapt.

6.2.1. Verificarea funcționalității

Pentru fiecare modul în parte s-au luat o serie de date de intrare corespunzătoare, s-au procesat cu ajutorul modulului și s-au urmărit rezultatele finale.

În partea de abrevieri s-au încercat operațiile disponibile asupra dicționarului și s-a constatat funcționarea acestora cu succes, pornind de la cele de adăugare și ștergere până la cea de căutare a unei abrevieri anume.

În partea de învățare, în primul rând a fost verificată corespondența între setul inițial de date și ceea ce se afișează în tabel. S-au testat o mulțime de valori posibile atât pentru coloanele setului de date cât și pentru tipul specific de etichetă, fiecare dintre acestea ajungând să arate ceea ce am așteptat. După acest pas, am testat partea de salvare a unei reguli selectate, în primul rând prin verificarea dacă s-a memorat în fișierul de ieșire care conține reguli, iar după confirmarea acestui lucru am mai verificat și dacă din setul inițial de date s-au șters acele linii care conțineau partea dreaptă a regulii selectate. Într-adevăr, fișierele se modificau conform așteptărilor. De asemenea, algoritmul Apriori de generare a seturilor frecvente se rula mereu pe datele obținute anterior, atâta timp cât nu schimbăm valorile box-urilor de sus, ceea ce a confirmat buna funcționare și a acestuia. În plus, s-au testat parametri pe care utilizatorul îi poate transmite algoritmului, asigurându-ne că se respectă limitele de suport și confidence.

În partea de adnotare s-a testat în primul rând înlocuirea abrevierilor. Întâi am ținut cont de un dicționar mai mic, pentru a fi mai ușor de urmărit prescurtările înlocuite.

După ce am văzut că acest lucru se realizează cu succes, am adăugat între două rulări succesive o prescurtare nouă în dicționar, care era deja în setul de date dar nu avea o înregistrare corespunzătoare în fișierul cu abrevieri. După adăugarea acestei noi prescurtări, am verificat dacă este și ea înlocuită în cadrul observațiilor, iar după confirmarea acestei operații am trecut mai departe la testarea modului de înlocuire al abrevierilor. În acest sens, am selectat o mică parte din datele pe care trebuie să se aplice procesarea, am selectat o serie de reguli și am urmărit rezultatele procesului de adnotare pentru valoarea la care se refereau acele reguli. Procesul a fost unul aproape instantaneu iar abrevierile unde se potriveau regulile au fost înlocuite cu succes.

6.2.2. *Validarea datelor generate*

Pentru validarea datelor generate, s-a urmărit la fiecare pas formatul fișierelor de ieșire generate de modulele aplicației. În principiu trebuiau să respecte formatul csv sau cele proprii – in și out - să nu lase rânduri sau spații libere unde nu era cazul și să fie consistente. În plus, o verificare mai amănunțită a fost făcută asupra fișierului final în care se genera adnotările pentru observațiile dorite, pentru a ne asigura că aplicația generează un rezultat final corect. S-a constatat că acesta respectă întru totul formatul așteptat, având o structura similară cu fișierele de la care se pornește învățarea.

6.3. Testare funcțională

6.3.1. *Impactul setului de date de învățare*

Această aplicație se bazează în primul rând pe un set de date variate și corect adnotate de pe urmă cărora să învețe reguli. Cu cât datele sunt mai complete, cu atât regulile generate sunt mai relevante și mai precise. Totuși, vrem să vedem cât de mare este impactul pe care aceste date de învățare îl au asupra procesului de generare de reguli, așadar am făcut o verificare pentru date de diferite dimensiuni, urmărind rezultatele obținute.

S-au ales 4 etichete la întâmplare pentru care să se facă aceste verificări, și s-a rulat procesul de învățare de regul pentru seturi de date de dimensiuni : 100, 250, 500, 750 și 1000, asta însemnând numărul de observații medicale adnotate disponibile. S-a urmărit numărul de reguli generate pentru fiecare dintre aceste cazuri, suportul fiind setat mereu la 30.

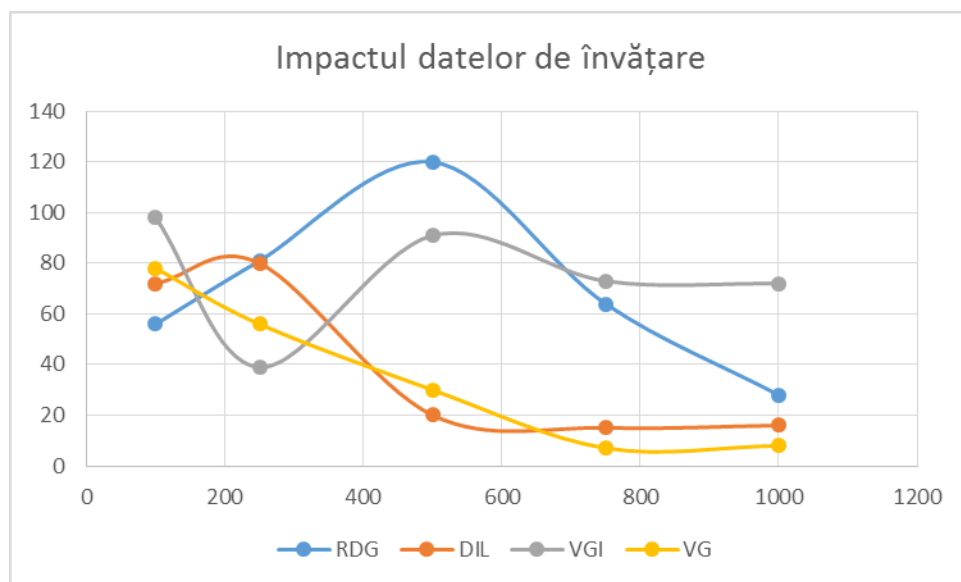


Figura 6.1 – Impactul datelor de învățare

După cum putem observa, la un număr mic de observații de pe urmă cărora să se învețe, numărul de reguli variază destul de mult. Totuși, când dimensiunea setului de învățare crește, pentru toate valorile de mai sus observăm o tendință de stabilitate, în sensul că se generează aproximativ același număr de reguli, care vor fi și de relevanță și precizie asemănătoare. Așadar, la un număr de înregistrări de peste 800 deja putem spune că generarea de reguli începe să se stabilizeze.

6.3.2. Impactul mai multor iterații ale algoritmului Apriori

Un alt aspect pe care am dorit să-l verificăm a fost impactul pe care îl au mai multe generări succesive de reguli asupra aceluiași valori, după ce utilizatorul memorează câte o regulă în fiecare pas. În urmă memorării, acea regulă se șterge și la fel se întâmplă și cu observațiile în care apar termenii cheie care generează regulă. Așadar, setul de date va scădea la fiecare pas, iar ceea ce vrem să urmărim este cum afectează acest lucru generarea de reguli.

Vom prezenta o situație în care pentru o etichetă utilizatorul execută 4 pași de tipul generare reguli + memorare o regulă, iar la fiecare etapă se vor afișa numărul de observații rămase și numărul de reguli generate.

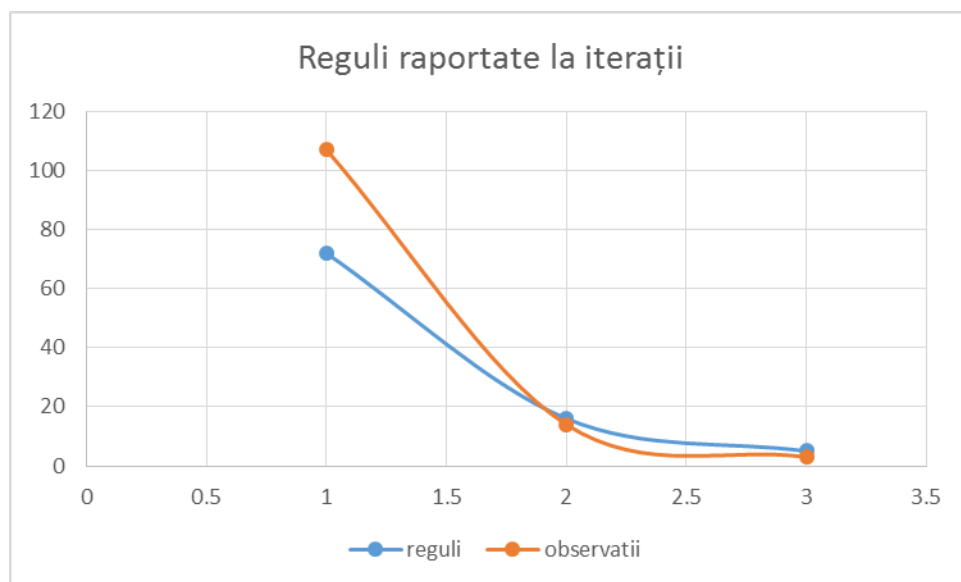


Figura 6.2 – Reguli raportate la iterații

Tendința care se observă este aceea că după reducerea setului de observații, numărul acestora tinde să fie egal cu numărul de reguli generate. Această corespondență de aproape 1 la 1 mai semnifică și faptul că regulile generate la urmă vor fi dintre cele mai precise, însă acestea vor fi generate pe datele rămase după ce utilizatorul a memorat deja ceea ce considera relevant.

6.3.3. Variații ale suportului

Unul dintre cele mai importante aspecte atunci când vine vorba de generarea de reguli se referă la valoarea selectată pentru suport. Practic, această ne impune o limită asupra unor reguli pe care algoritmul le consideră mai puțin frecvente, iar acele reguli care sunt sub limită nici măcar nu mai sunt afișate utilizatorului. În mod normal, regulile cele mai bune sunt acelea cu suportul cel mai mare, însă fiind vorba de limbaj natural, nu întotdeauna se vor selecta cele mai relevante și mai expresive cuvinte de la care să se deducă o adnotare.

Așadar, trebuie să descoperim cam care ar fi cea mai potrivită valoare pe care utilizatorul să o aleagă, cel puțin pentru primele iterații ale algoritmului de generare. În acest sens, am considerat că un test relevant ar fi acela în care să luăm mai multe etichete la întâmplare, și să generăm pentru ele reguli pentru cât mai multe variante de suport. În felul acesta, vom putea urmări numărul de reguli generate pentru fiecare dintre aceste valori, căutând o oarecare stabilitate.

Am ales astfel 3 etichete în mod aleator, și am generat reguli pentru ele setând suportul la valorile: 10, 15, 20, 25, 30, 35, 40, 50, 60, 70.

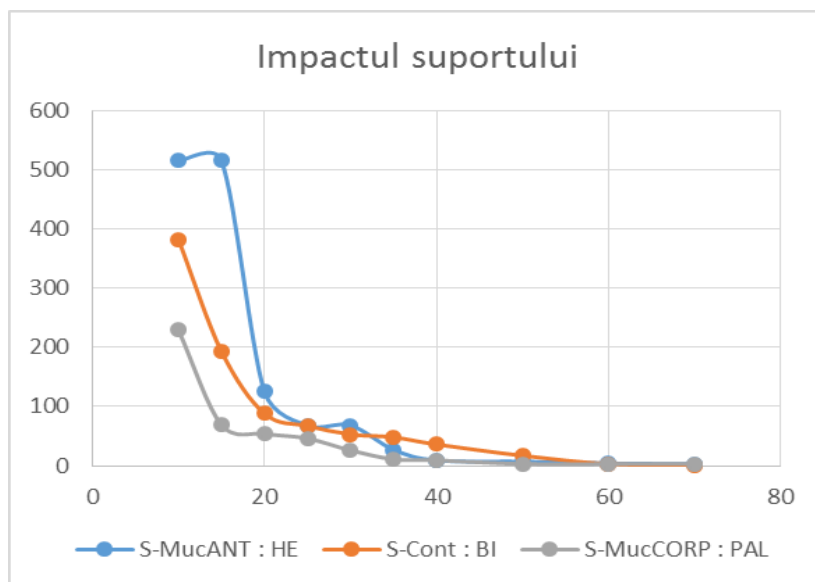


Figura 6.3 – Variații ale suportului

Rezultatele acestui grafic par a fi foarte încurajatoare. Pentru toate cele 3 valori alese, observăm că de la un moment dat numărul de reguli generate tinde să fie tot mai stabil. În plus, observăm că această stabilitate apare în toate cele 3 cazuri aproximativ în jurul aceleiași valori, așadar putem deduce faptul că am putea găsi o valoare a suportului care să fie potrivită pentru orice tip de eticheta, cel puțin în primele iterații ale generării de reguli. Din câte putem deduce în urmă acestor experimente, acea valoare universal potrivită ar fi undeva între 25 și 30.

6.3.4. Testarea înlocuirii abrevierilor

Un ultim scenariu de test pe care am dorit să îl realizăm este acela cu privire la înlocuirea abrevierilor. Așa cum am mai specificat și până acum în cadrul acestei lucrări, observațiile medicale, fiind scrise în limbaj natural și conținând o serie de termenei prescurtați, trebuie să fie trecute printr-un proces de înlocuire al abrevierilor înainte de a putea fi procesate.

Corectitudinea acestui proces are un impact major asupra rezultatelor finale, întrucât regulile se aplică în mod corespunzător doar pe textul în formatul complet. Astfel, procesul de înlocuire trebuie să aibă o rată cât mai mare de succes, atât în privința numărului de cuvinte detectate cât și în privința corectitudinii cu care acestea sunt detectate.

Testul pe care am dorit să îl efectuăm este compararea setului de date de adnotat în forma inițială, adică în forma în care conține prescurtări, cu același set de date după trecerea prin procesul de înlocuire a abrevierilor. Metrica pe care am folosit-o în cadrul acestei comparații este numărul total de cuvinte din toate observațiile puse împreună. Așadar, s-a constatat că în forma inițială, cele aproape 1000 de observații medicale ce trebuie adnotate conțin în total 7978 de cuvinte, iar în urmă înlocuirii abrevierilor am obținut 9847 de cuvinte. Desigur, această diferență depinde de mai mulți factori precum lungimea unei abrevieri sau numărul de abrevieri compuse, însă ne ajută să avem o idee despre cât de mult se extinde textul după înlocuire. Chiar mai mult, ne poate ajuta să ne

dăm seama și de densitatea abrevierilor din cadrul acestor observații medicale, de exemplu în cazul de față avem o creștere de 2000 de cuvinte în cadrul a 1000 de observații medicale, ceea ce ar putea sugera un număr de aproximativ două abrevieri pentru fiecare observație.

Capitolul 7. Manual de Instalare si Utilizare

În cadrul acestui capitol, se vor detalia instrucțiunile de instalare ale sistemului, cât și modul de utilizare a acestuia din punct de vedere al utilizatorului.

7.1. Manual de instalare

Instalarea acestui sistem, este una foarte simplă deoarece este implementată într-un limbaj de programare orientat obiect și ușor de învățat.

Fiind o aplicație implementată utilizând limbajul Java 8, pentru rularea aplicației este nevoie să fie instalat pe sistemul de operare JDK – Java Development Kit. Acest mediu poate fi descărcat de pe diferite site-uri prin introducerea cuvintelor cheie (exemplu: jdk download) în browser. Un exemplu de site de unde se poate descărca această platformă este:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html> . În funcție de sistemul de operare pe care rulează calculatorul se va descărca versiunea corespunzătoare. După descărcarea platformei JDK se va lansa în execuție fișierul executabil și se vor urma pașii de instalare care vor fi afișați pe ecran. Aceștia includ alegerea locației de instalare și alegerea facilităților care vor fi instalate.

Cu toate că la dezvoltarea aplicației a fost folosit utilitarul Netbeans, nu este necesară instalarea lui pe sistemul pe care se dorește rularea aplicației.

7.2. Manual de utilizare

După ce s-au executat cu succes pașii de instalare de la secțiunea anterioară, aplicația ar trebui să poată fi folosită cu succes. Lansarea ei în execuție se va face prin intermediul unui fișier executabil cu extensia .jar. Acest fișier este localizat în folderul dist din cadrul directorului în care se află aplicația. Pentru utilizarea proiectului se va face un dublu-click pe fișierul executabil. După efetuarea acestui lucru aplicația este lansată, interfața ei de pornire fiind cea din figura 7.1.

Interfața acestei aplicații este simplă de folosit deoarece numele butoanelor, tabelelor fiind foarte sugestie.

Interfața de start a aplicației prezintă o fereastră cu butoane de legătură către cele trei mari module disponibile, precum și un buton de închidere a acesteia.

Fiecare buton afișat în interfață deschide o nouă fereastră care prezintă interfața grafică a modului respectiv:

Butonul “Annotate Data” - deschide interfața modulul de adnotare.

Butonul “Generate Annotation Rules” – deschide interfața modulul de învățare și generare de reguli.

Butonul “AbbreviationsEditing” – deschide interfața modulul de editare a abrevierilor.

Butonul “Exit” – închide aplicația.

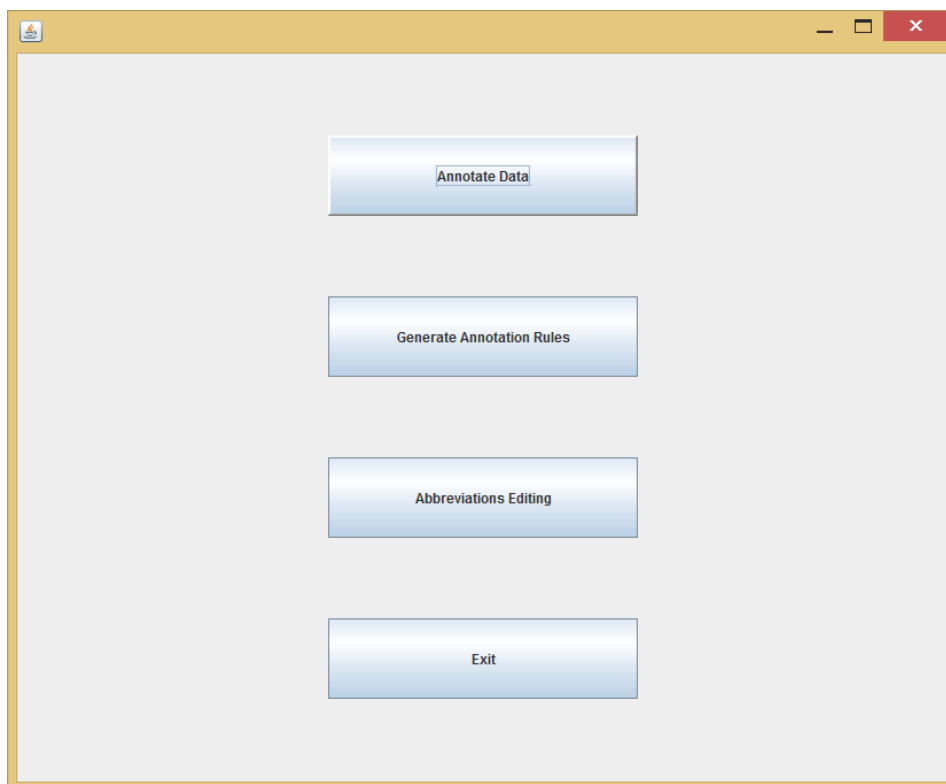


Figura 7.1 Interfața de start a aplicației

7.2.1. Interfața modului de adăugare și actualizare a abrevierilor

Interfața acestui modul este destinată operațiilor asupra abrevierilor. După cum se poate observa și în figura 7.2 sunt afișate într-un tabel perechi de forma „Abbreviations – Significance” care cuprind pe fiecare linie abrevierea și semnificația acesteia. Datele din tabel pot fi sortate crescător sau descrescător în funcție de preferințele utilizatorului dând click pe header-ul tabelului. Când săgeata este orientată în sus datele sunt sortate crescător, iar când săgeata este orientată în jos datele sunt sortate descrescător.

În partea dreaptă sus a ferestrei este afișat butonul de „Find word” care caută în tabel abrevierea introdusă în căsuța de deasupra lui. Cu ajutorul butoanelor de „Save” „Delete”, „Add row” se poate efectua operațiile de:

- salvarea modificărilor asupra înregistrărilor din setul de abrevieri
- ștergerea unei înregistrări din setul de abrevieri
- adăugarea unei noi înregistrări în setul de abrevieri.

Butonul „Back” este utilizat pentru revenirea la pagina principală a aplicației.

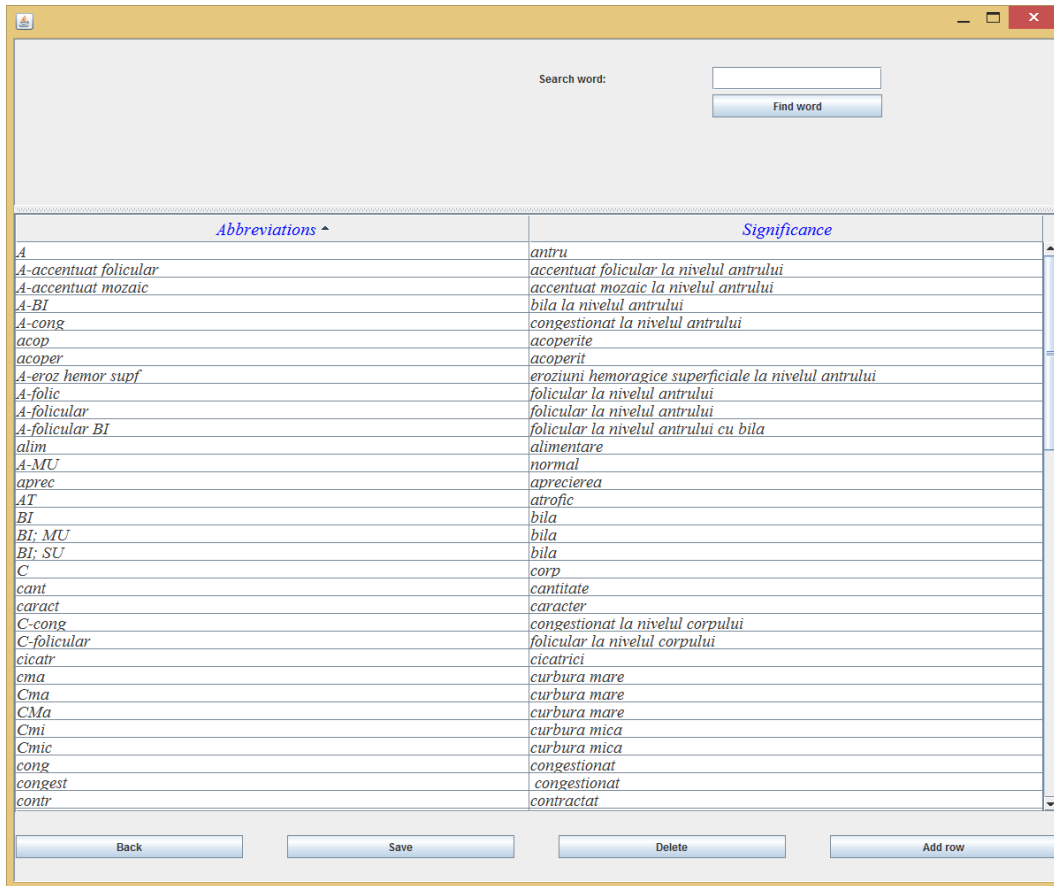


Figura 7.2 Interfața modulului de adăugare și actualizare a abrevierilor

7.2.2. Interfața modulului de învățare și generare de reguli

Această interfață prezintă operațiile care pot fi efectuate în cadrul modulului de învățare și generare de reguli.

Interfața pune la dispoziția utilizatorului posibilitatea de a alege setul de date pentru care să genereze reguli. Pașii pentru generarea și validarea regulilor sunt următorii:

- 1) Selectarea din combobox a unei valori care reprezintă numele coloanei și apăsarea butonul “Get column values”.
- 2) După efectuarea pasului anterior, combobox-ul din partea dreaptă o să fie populat cu valorile posibile pentru coloana aleasă. Utilizatorul trebuie să aleagă și din acest combobox valoarea dorită.
- 3) Dacă s-au ales valorile dorite, se va da click pe butonul “Generate rules” pentru a genera noi reguli. O dată la apăsarea acestui buton, se va deschide o nouă fereastră de unde trebuie setați parametrii algoritmului Apriori. După setarea parametrilor la apăsarea butonului “OK” tabelul va fi populat cu regulile generate.
- 4) Reguli generate pot fi selectate de către utilizator iar la apăsare butonului “Drop covered lines and save rules” se efectuează două operații simultan
 - Salvarea regulilor selectate pentru a putea fi folosite apoi în cadrul procesului de adnotare

- Ștergerea din setul de date exemplele care sunt acoperite de aceste reguli

Figura 7.3 Fereastra care se deschide la click-ul butonului Generare Rules pentru setarea parametrilor ai algoritmului Apriori.

Rules	Support	Select
S-Lum RDGE <- duodeno	100	<input type="checkbox"/>
S-Lum RDGE <- duodeno esofagian	100	<input type="checkbox"/>
S-Lum RDGE <- duodeno gastro	100	<input type="checkbox"/>
S-Lum RDGE <- duodeno gastro esofagian	100	<input type="checkbox"/>
S-Lum RDGE <- esofagian	100	<input type="checkbox"/>
S-Lum RDGE <- gastro	100	<input type="checkbox"/>
S-Lum RDGE <- gastro esofagian	100	<input type="checkbox"/>
S-Lum RDGE <- reflux	100	<input type="checkbox"/>
S-Lum RDGE <- reflux duodeno	100	<input type="checkbox"/>
S-Lum RDGE <- reflux duodeno esofagian	100	<input type="checkbox"/>
S-Lum RDGE <- reflux duodeno gastro	100	<input type="checkbox"/>
S-Lum RDGE <- reflux duodeno gastro esofagian	100	<input type="checkbox"/>
S-Lum RDGE <- reflux esofagian	100	<input type="checkbox"/>
S-Lum RDGE <- reflux gastro	100	<input type="checkbox"/>
S-Lum RDGE <- reflux gastro esofagian	100	<input type="checkbox"/>

Figura 7.4 Interfața modului de învățare și generare reguli

7.2.3. Interfața modului de adnotare

Interfața acestui modul este prezentată în următoarea figură.

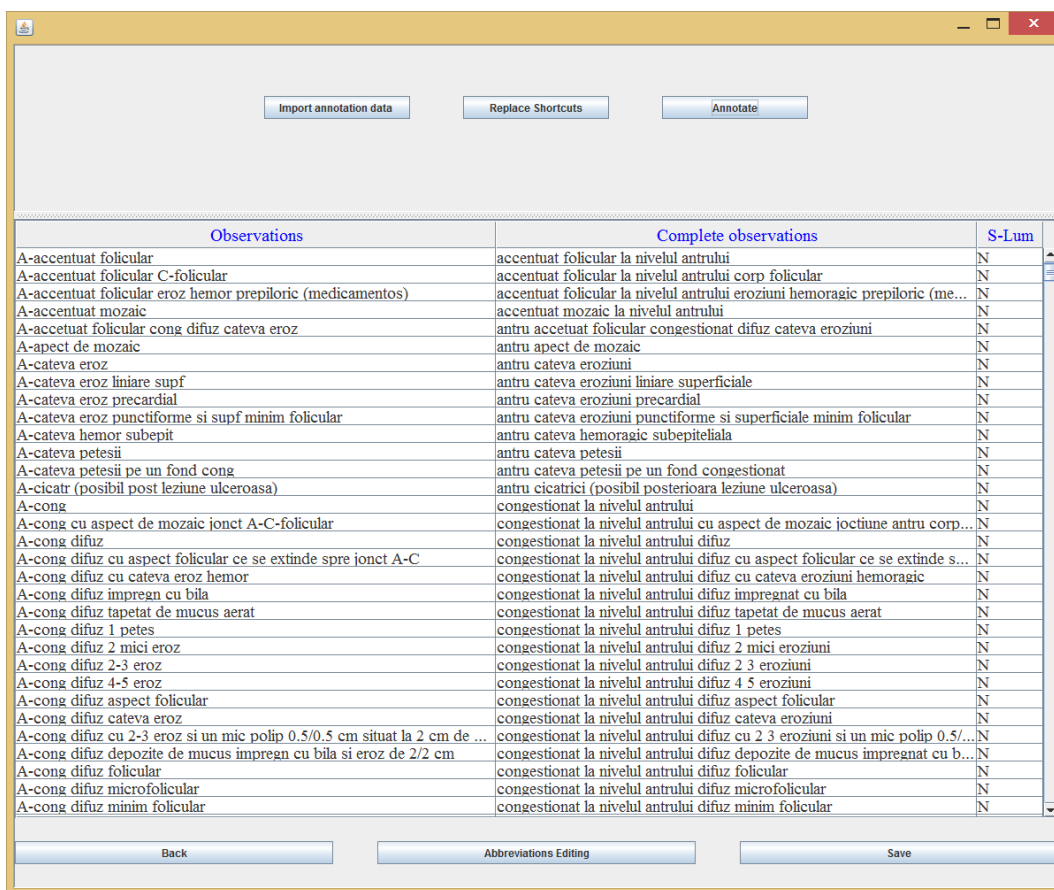


Figura 7.5 Interfața modului de adnotare

La fel ca și celelalte interfețe grafice prezentate mai sus, și această interfață este una simplistă și ușor de folosit. În mijlocul ferestrei este afișat un tabel care conține pe prima coloană observațiile în forma normală care conține abrevieri. Prima coloană se încarcă în tabel la apăsarea butonului „Import annotation data” care oferă utilizatorului de a selecta un fișier .csv pentru care vrea să facă adnotarea. A doua coloană se populează la click-ul butonului „Replace Shortcuts” în urma căruia se înlocuiesc abrevierile. Pentru adnotarea datelor se va executa un click pe butonul „Annotate” care în funcție de coloana aleasă va completa o nouă coloană cu etichetele generate. La fel ca și la celelalte interfețe butonul de „Back” - revine la pagina de start iar butonul „Save” – salvează datele într-un fișier. Butonul „Abbreviations Editing” este o scurtătură spre interfața modului de abreviere.

Capitolul 8. Concluzii

Extragerea de informații din texte scrise în limbaj natural și organizarea acestora într-o formă structurată reprezintă în continuare un domeniu de cercetare important și foarte studiat. Această operație nu este deloc una simplă, mai ales atunci când vine vorba de date din domeniul medical, care conține o varietate de termeni specifici, definiții și notații speciale.

Această aplicație și-a propus să ofere o soluție performanță și utilă în acest sens, bazându-se pe o serie de concepte și tehnici mai puțin întâlnite în cazul altor aplicații asemănătoare. Modul de funcționare al acesteia a fost implementat pentru cazul specific al examinării datelor endoscopice, însă se poate adapta într-o manieră similară și pentru alte tipuri de observații medicale. De asemenea, aplicația a fost realizată pentru a interpreta text în limba română, ceea ce puține sisteme similare realizează, majoritatea fiind optimizate pentru limbi mai populare precum engleză.

Sistemul realizat a demonstrat că parsarea lingvistică nu este neapărat singură tehnică ce se poate aborda în extragerea de informații din text nestructurat. De asemenea, a mai arătat faptul că limba română poate fi și ea procesată chiar și atunci când vine vorba de observații medicale, care conțin termeni mai specifici și mai rar întâlniți în vocabularul obișnuit. Nu în ultimul rând, aplicația a mai prezentat o nouă utilitate a algoritmului Apriori, care generează mulțimi de elemente frecvente și care e folosit cu precădere în analiza coșurilor de cumpărături. Acesta s-a dovedit a fi o alegere excelentă în găsirea elementelor frecvente din cadrul observațiilor medicale și a fost un principal punct de sprijin al aplicației în general.

8.1.1. Rezultate obținute

În continuare vom urmări rezultatele obținute în urmă dezvoltării aplicației și modul în care acestea corespund cu cerințele stabilite inițial.

Așadar, o prima cerință era extragerea automată de informații din cadrul textelor în limbaj natural obținute în urmă endoscopiilor. Așa cum am specificat și mai sus și după cum am putut observa pe parcursul lucrării, în special în partea de testare și validare, generarea de adnotări pentru observații medicale se realizează cu succes și cu un grad mare de precizie. Această precizie depinde și de modul în care utilizatorul alege să salveze regulile de adnotare, însă fiind vorba de un expert uman în domeniu, putem considera că acesta va ști să aleagă regulile mai relevante, iar asta va duce la o adnotare corectă și precisă.

A doua cerință specificată se referea la modul de generare al regulilor, astfel că acesta să fie unul cât mai eficient. Prin folosirea algoritmului Apriori și posibilitatea de a schimba parametri de intrare ai acestuia, ne-am asigurat că utilizatorul va avea flexibilitate în ceea ce privește generarea automată a regulilor, setând acei parametri în funcție de cum arată setul de date. În urmă efectuării unor experimente prezentate și în capitolul 6, am stabilit că o valoare a suportului undeva între 25 și 30 ar fi cea mai potrivită pentru majoritatea cazurilor, asigurând astfel generarea din start a unor reguli potrivite.

O altă cerință importantă pe care o are sistemul este înlocuirea cât mai corectă a abrevierilor din cadrul observațiilor medicale. Și asupra acestui aspect am realizat

anumite experimente, constatând atât corectitudinea înlocuirilor cât și gradul de acoperire al mulțimii de abrevieri care apar în observațiile endoscopice. Ambele aspecte s-au dovedit a fi îndeplinite cu succes, ceea ce confirmă și respectarea acestei cerințe.

Nu în ultimul rând, s-a dorit ca această aplicație să aibă o interfață prietenoasă prin intermediul căreia utilizatorii să poată interacționa ușor cu ea, fără a fi nevoie de cunoștințe suplimentare din domeniul informatic. Într-adevăr, acest lucru a fost și el îndeplinit prin modul în care a fost creată și organizată interfața grafică a sistemului. Modulele principale sunt clar delimitate, iar în interiorul fiecăruia dintre ele sunt bine definite operațiile care se pot efectua, fiecare dintre acestea fiind denumită într-un mod sugestiv.

Aceste aspecte fiind prezentate, consider că aplicația dezvoltată a reușit să își atingă scopul și să prindă o soluție foarte bună la problema stabilită inițial.

8.1.2. Dezvoltări ulterioare

Pentru dezvoltarea ulterioară a acestei aplicații, se pot realiza o serie de îmbunătățiri care vor fi prezentate în ceea ce urmează. Acestea se pot referi, pe de-o parte, la performanța generală a aplicației, iar prin asta ne referim atât la precizia ei cât și la viteză de execuție a operațiilor. Pe de altă parte, se pot aduce îmbunătățiri și pe partea funcțională, în sensul că i se mai pot adăuga funcționalități suplimentare, atât pentru cazurile endoscopiilor cât și pentru alte domenii medicale similare.

Pentru început, pe partea de viteză de execuție, o îmbunătățire ar putea fi adusă modulului de generare de adnotări, atât operației de înlocuire a abrevierilor cât și celei de generare a etichetelor. În momentul de față, se parcurg observațiile medicale în mod secvențial și se verifică prezența abrevierilor în cadrul acestora. La fel se procedează și în cazul adnotării, la verificarea regulilor. O idee de a grăbi acest proces este aceea de a face procesul să fie multithreaded. Acest lucru s-ar putea realiza prin împărțirea setului de date care trebuie adnotat în mai multe părți distincte, în funcție de numărul optim de threaduri. Fiecare astfel de thread va procesa partea să de observații și se va ocupa în prima fază de înlocuirea abrevierilor din cadrul acesteia. Pentru a nu crea conflicte de acces concurrent, se vor crea liste distincte de observații pentru fiecare fir de execuție iar apoi în final rezultatele fiecărui thread se vor pune în comun pentru a genera nouă lista cu abrevierile înlocuite. Într-o manieră similară se poate face și adnotarea observațiilor, prin această împărțire în părți distincte. Acest proces ar fi de mare ajutor, întrucât seturile de date de adnotat sunt de obicei foarte mari, de ordinul miilor de înregistrări, iar creșterea de performanță în acest caz ar fi foarte vizibilă și semnificativă.

Pe lângă partea de viteză de execuție, o îmbunătățire am putea aduce tot procesului de adnotare, la verificarea potrivirii cu regulile generate în cadrul procesului de învățare. Îmbunătățirea ar consta în utilizarea JAPE (Java Annotation Patterns Engine). Acesta este o funcționalitate a sistemului GATE care permite recunoașterea expresiilor regulate în cadrul adnotării unor documente, modelul acesteia fiind bazat pe grafuri, în acest fel aplicându-se expresiile regulate pe structuri de date mai complexe decât simple Strîng-uri. În acest fel, precizia potrivirilor dintre reguli și observațiile medicale ar crește, iar eficiența totală a procesului de adnotare ar crește odată cu ea.

Că și îmbunătățiri pe partea funcțională, am putea implementa noi seturi de module care să se ocupe și cu alte tipuri de observații medicale. Date nestructurate în limbaj natural sunt prezente peste tot în domeniul medical, așadar așa cum s-a realizat

această adaptare pentru endoscopii, s-ar putea realiza și pentru alte tipuri de investigații. Maniera de generare de reguli poate fi similară, căutând seturi de cuvinte frecvente în cadrul observațiilor medicale, și în mod asemănător se poate realiza și adnotarea datelor. Singurul lucru care ar fi mai diferit ar fi dicționarul de abrevieri, fiind nevoie de altfel de termeni, însă acesta nu reprezintă un aspect dificil de rezolvat.

Bibliografie

- [1] Louise Deleger, Cyril Grouin, “Extracting medical information from narrative patient records: the case of medication-related information” ,
- [2] Stuart Russel, Peter Norvig, “Artificial intelligence, a Modern Approach”
- [3] Udo Hahn, Martin Romacker, “Text Structures in Medical Text Processing: Empirical Evidence and a Text Understanding Prototype”
- [4] Ole Kristian Fivelstad, “Temporal Text Mining”
- [5] http://en.wikipedia.org/wiki/Machine_learning
- [6] <http://www3.cs.stonybrook.edu/~cse634/presentations/TextMining.pdf>
- [7] <http://rochi.utcluj.ro/rrioc/articole/RRIOC-7-1-Florea.pdf>
- [8] <http://www3.cs.stonybrook.edu/~cse634/presentations/TextMining.pdf>
- [9] <http://documents.software.dell.com/Statistics/Textbook/data-mining-techniques#mining>
- [10] <http://searchcontentmanagement.techtarget.com/definition/natural-language-processing-NLP>
- [11] <http://rochi.utcluj.ro/rrioc/articole/RRIOC-2012-3-Cercel.pdf>
- [12] http://link.springer.com/chapter/10.1007%2F978-1-4757-3873-5_21#page-1

Anexa (1)

Lucrări publicate

Paulina Mitrea

To

al3xandra.bali@yahoo.com

CC

Paulina Mitrea

Jun 17 at 9:15 AM

Buna ziua,

Am deosebita placere de a va aduce la cunostinta faptul ca articolele Dv. transmise spre evaluare pentru "Computer Science Student Conference 2015", au fost acceptate pentru a fi prezentate in cadrul conferintei. Va rog sa transmiteti aceasta instiintare si coordonatorilor Dv.

Va rog sa-mi confirmati primirea acestui email. In caz ca aveti nevoie de detalii suplimentare, va rog sa ma contactati neintarziat pe unul dintre numerele mele de mobil, specificate mai jos.

Conf. Dr. Paulina Mitrea

Learning Annotation Rules for Natural Language Texts

Alexandra Bali

Department of Computer Science
Technical University of Cluj-Napoca
Barițiu 28, RO-400027 Cluj-Napoca, Romania
Alexandra.Bali@student.utcluj.ro

Abstract—

In order to automatically analyze medical observations obtained by endoscopies, an individual needs to collect relevant information and organize it in such a way that it would become processable. To achieve this, we present an extraction module that is capable of processing the natural language written down after these kind of medical observations. What this module does different compared to other similar applications is that it does not need to linguistically parse the natural language. Instead, in the first step, it analyzes the observations and generates rules based on them. After the user confirms the correctness of the rules, the second step begins and the application creates annotations by applying the rules on the natural language. This article presents a case-study based on a big list of medical observations, used as input data, it explains the process of creating rules and annotations and finally it discusses the correctness of the results and compares them to other related work.

I. INTRODUCTION

The influence of computer science in medicine has been increasing significantly in the recent years, as medical data processing became a research trend which has been given high priority. Classifying and analyzing patient data by all possible means can enable the improvement of decision-making when it comes to giving a diagnosis. There are various ways to reuse the information stored in the electronic health records, the main purpose being the automatic processing of the natural text paragraphs which contain the most relevant observations and findings. This kind of processing involves extracting significant information from the records, which is subject to many efforts for some years. The need of such information extraction has made computer science specialists design many successful theories and systems that are able to process natural language in different ways, each of them having their own advantages and disadvantages.

In the case of endoscopies, by examining the health records we can see that they usually consist in brief texts in natural language, also containing a large amount of abbreviations. This writing style makes it difficult to implement an automatized system that could process/analyze such records. First of all, the aim would be to examine this kind of medical observations and create a series of surveys that would be useful in the process of medical diagnosis. By this, several diseases will be grouped

together such that one/someone will be able to observe which ones are related, for example. To be able to do this, we need structured data that can be classified (by some categories) in an automatic way. Although for this kind of problem a solution could be the linguistic parsing of the observations, we present a system that is able to generate annotations just by parsing the texts, examining frequent rules and applying them back on the medical records. The key of the system is that it is able to learn what kind of rules it should apply, and to keep them up to date for future usage. The updating of the rules can be done manually by a specialist, but the generation is done automatically by using the Apriori algorithm.

This kind of approach has a series of advantages over linguistic parsing applications, mainly because there is no need for a parser to interpret the meaning of the records. In this way, the pre-processing step in which the text would be sophisticatedly analyzed is not required anymore. In addition, no part-of-speech tagger or lexicon is needed. Our approach is based on the usage of the Apriori algorithm, which is able to generate frequent item sets and association rules over transactional databases. Prior to applying the algorithm, a parsing of the medical records need to be done in order to replace the contained abbreviations, as these records are often written in brief text. After the rule generation, a human expert has the ability to update the rules, deleting the ones that are not relevant (for a particular case). Finally, the annotation of the text will take place by applying the updated rules on the initial medical texts and filling up for every record the desired columns that represent observed parts with the corresponding values of the observations. In this way, the user will have well-structured information about a big number of records, being able to further examine them and discover all kinds of associations. The presented system is the first Apriori-based approach for this type of texts in Romanian, and the results confirm that it performs quite well.

The paper is organized as follows. Section II presents the Apriori algorithm in more detail, part A containing general information about rule induction using Apriori and details about the definition of the parameters. Part B will describe the approach used in designing our system and the way in which the algorithm is used for text annotation. Section III

presents the system architecture, the modules involved along their functionalities, the flow of the application and a series of screenshots. After that, Section IV will discuss the obtained results for different ways of running the application. This includes variations of the values for support and confidence for Apriori, the correctness of the shortcut replacing and the completeness of the final annotation results. Several experiments will be presented in order to calibrate the system for high precision. Section V compares our application to other similar existing approaches, which will be detailed at the time, and finally Section VI will contain the conclusions of this paper and future work.

II. EMPLOYING APRIORI FOR ANNOTATING NATURAL LANGUAGE TEXTS

A. Rule induction using Apriori

In data mining, association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. Piatetsky-Shapiro describes analyzing and presenting strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, Agrawal introduced association rules for discovering regularities between products in a large scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule $\{\text{onion, potatoes}\} \Rightarrow \{\text{burger}\}$ found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, he or she is likely to also buy burger. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection and bioinformatics. In computer science and data mining, Apriori is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing).

Definition: Following the original definition by Agrawal the problem of association rule mining is defined as: Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of n binary attributes called items. Let $D = \{t_1, t_2, \dots, t_n\}$ be a set of transactions called the database. Each transaction in D has a unique transaction ID and contains a subset of the items in I . A rule is defined as an implication of the form $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The sets of items (for short itemsets) X and Y are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively. To illustrate the concepts, we use a small example from the supermarket domain. The set of items is $I = \{\text{milk, bread, butter, beer}\}$ and a small database containing the items (1 codes presence and 0 absence of an item in a transaction) is shown in the table below. An example rule for

the supermarket could be $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ meaning that if milk and bread is bought, customers also buy butter.

Note: this example is extremely small. In practical applications, a rule needs a support of several hundred transactions before it can be considered statistically significant, and datasets often contain thousands or millions of transactions.

Transaction ID	Milk	Bread	Butter	Beer
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0
6	1	0	0	0
7	0	1	1	1
8	1	1	1	1
9	0	1	0	1
10	1	1	0	0
11	1	0	0	0
12	0	0	0	1
13	1	1	1	0
14	1	0	1	0
15	1	1	1	1

To select interesting rules from the set of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence.

Support

The support $\text{supp}(X)$ of an itemset X is defined as the proportion of transactions in the data set which contain the itemset.

$$\text{supp}(X) = \text{no. of transactions which contain the itemset } X / \text{total no of transactions}$$

In the example database, the itemset $\{\text{milk, bread, butter}\}$ has a support of $4/15 = 0.26$ since it occurs in 26% of all transactions. To be even more explicit we can point out that 4 is the number of transactions from the database which contain the itemset $\{\text{milk, bread, butter}\}$ while 15 represents the total number of transactions.

Confidence

The confidence of a rule is defined:

$$\text{conf}(X \rightarrow Y) = \text{supp}(X \cup Y) / \text{supp}(X)$$

For the rule $\{\text{milk, bread}\} \Rightarrow \{\text{butter}\}$ we have the following confidence: $\text{supp}(\{\text{milk, bread, butter}\}) / \text{supp}(\{\text{milk, bread}\}) = 0.26/0.4 = 0.65$

This means that for 65% of the transactions containing milk and bread the rule is correct. Confidence can be interpreted as an estimate of the probability $P(Y|X)$, the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS.

B. Apriori for text annotation

Apart from the applications presented in section A, we tried to use the Apriori algorithm for a new kind of approach: parsing medical records obtained from endoscopies. The idea appeared after observing the resemblance between several records, and the repetitive nuance of the diagnoses. As Apriori is used for frequent item set mining, it seemed like it could give good results in the process of information extracting from large sets of medical records. The algorithm is applied after the abbreviation replacement, during the step where we have the observations in the complete form. After parsing the entire data set, it will determine in the first step the frequent sets of diagnoses, and after this, based on these sets, it will generate association rules which will highlight general trends that are present in the medical records.

First of all, before we can use the Apriori algorithm on any set of observations, a series of basic rules must be generated so that we will have a starting knowledge base. In time, this knowledge base will expand and it will become more and more consistent, but for the beginning it does not need to be very large. In order to obtain this initial knowledge base, we will use an already existing database of medical records that contains annotations for different examined parts during an endoscopy. The database is organized as it follows: every diagnosis written in natural language has a set of corresponding annotations for the examined parts, each of these having a different meaning. The algorithm will be applied sequentially on each part, and it will generate the set of rules that corresponds to the observations made on that part. The rules will be presented in the graphical interface of the application, so that an expert could analyze them and delete what he considers to be not relevant. After this step, our system will have a starting knowledge base for every part that was examined, which consists in a set of rules. An example of such a rule could be: for the "stomach lumen" part and "normal" value, we could have: $\{S-Lum_N \rightarrow \text{follicular}\}$. Using these rules, we can get to the next step where the system has the ability to apply them on other medical records. This time, there is no need for the records to be annotated, because the application will automatically do this, which is actually its final purpose. In order to check the correctness of the annotation process, the generated rules could be applied on a copy of the initial data, from which we started to build the knowledge base. In this way, our results could be compared to the manual written results, this enabling us to see two different things: firstly, the correctness of the system will be tested, because we will have the correct results and the generated results, which could be compared. Secondly, we will be able

to adjust the parameters of the algorithm in several ways to see which one generates the best results.

For example, when creating the rules, running Apriori on a desired observed part after setting a big support will result in very few rules, as we explained in section A the role of the support parameter. This could have a positive impact on the precision of the system, meaning that if a diagnosis of a stomach part or property is given an annotation, there is a very big chance that the annotation is correct. On the contrary, the number of specifically generated annotations will be lower, and many observations will be given the default value. On the other hand, if the support is set to a much lower value, the number of created rules will increase significantly. This will affect the annotation process in the opposite way as described above: there will be more specific annotations, as the matching will be made easier, but the precision will decrease. The calibration process needs to be done carefully, but we will discuss more about this in Section IV. Until then, Section III presents the architecture of the system describing the modules and their functionality.

III. SYSTEM ARCHITECTURE

The annotation system is built on the Model-View-Controller architectural pattern, being implemented in a way that separates the graphical interface from the application logic. When it is launched in the first place, the application presents its main screen that consists in 3 buttons that lead to subsections and an Exit button. The first subsection is named Annotate Data, and it represents the main part of the system. The second one leads to the Generate Annotation Rules module, which will allow the user to create, update and delete the rules for a desired examined part and a desired value of the examination. Finally, the third section leads to the Abbreviation Editing part of the system, where we can manage the list of abbreviations that is used in the first step of running the program/application.

Before describing the modules, we need to say that the flow of the program, looking at the graphical interface, starts from the bottom to the top. The first thing that one needs to do in order to use the system is to check the meaning of the abbreviations. After confirming that this part is well formed, the next thing to do is to generate rules for what the user desires, rules that will be used in the annotation process. The newly created rules are shown to the user, which may select the irrelevant ones and delete them, or create other ones again by changing the parameters. Moving on, the third and final step consists in using the Annotate Data module, which will be generating the corresponding values for each medical record. Each of these three sections will be detailed below.

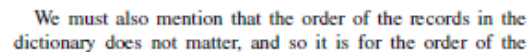
The Abbreviation Editing module presents a simple interface that allows the user to manage the abbreviations. As we described in the beginning of this article, medical observations consist in brief texts written in natural language, using a variety of short words and abbreviations. In order to correctly

to add the desired columns for which the annotations will be created. The creation will be made based on the existing rules, comparing the medical records with them so that the matching annotation should be written. Below the table there is a shortcut to the Abbreviations Editing module, in case the user needs to quickly change something, and a save option to memorize the results.

Generate Annotation Rules section is the next one in the flow. First of all, we must say that this section does not necessarily need to be accessed during every run of the program. It actually refers to the learning process of the system, and not specifically to the annotating part. The annotation process needs a set of rules that would be applied on the medical records, but once these rules are generated, they can be used as many times as the users wants to. Of course, our system must continuously improve itself, and this is done within the current section. Similarly to the previous one, a table occupies the middle of the window, in which the newly created rules will be shown.

IV. RESULTS

Finally, the last but not the least module of the system is Annotate Data. Actually, this is where the most important part of the application happens where the annotations are generated for all the medical records. We assume that until this step the user has updated the abbreviations dictionary and that he has a set of rules that are ready to be applied. The center of this application module presents the well-known table from the last sections, which will contain the observations both in the short and long form, after the abbreviations will be replaced. Beside these columns, the user has the ability to select the parts for which he wants to create annotations, and in this way new columns will be added, columns that will contain the corresponding abbreviations. Above the table there are the commands to import the medical data, to replace the shortcuts from this data using the dictionary and finally



support of the others seems to decrease, and in this case the user should also decrease the desired support when generating again new rules, after the deletion.

Apriori parameters

Select the columns name:

Select the columns value:

Confidence:

Support:

Max items:

	Rule	Support	Confidence
1	L item \rightarrow M item	0.0017	0.9
2	L item \rightarrow M item	0.0017	0.9
3	L item \rightarrow M item	0.0017	0.9
4	L item \rightarrow M item	0.0017	0.9
5	L item \rightarrow M item	0.0017	0.9
6	L item \rightarrow M item	0.0017	0.9
7	L item \rightarrow M item	0.0017	0.9
8	L item \rightarrow M item	0.0017	0.9
9	L item \rightarrow M item	0.0017	0.9
10	L item \rightarrow M item	0.0017	0.9

The experiments have shown that a support value around 30 would be a good starting point at the first rule creation, followed by a slightly decrease if some of the rules are deleted

V. RELATED WORK

Taking a look at some related work, we can see that there are not many similar approaches yet. The most significant related article seems to be Mining Information Extraction Rules from Datasets Without Linguistic Parsing. This approach focuses on a hybrid approach based on association rules mining and decision tree learning that does not require any linguistic parsing, contrary to sever information extraction systems based on machine learning techniques. The system presented in this article can also be parameterized in various ways, like ours, in order to influence the efficiency of the information extraction. Their experiments have shown that the system does not need a large training set to perform well.

VI. CONCLUSIONS AND FUTURE ORK

Our presented system for medical texts annotation proved to be an efficient tool that performs well on a variety of medical records. The learning process is simple and fast, even though it needs to start from a set of already annotated observations, the knowledge base increases rapidly and, as a consequence, the overall performance of the system becomes higher. The Apriori algorithm was suitable in the process of rules generation, and the overall behavior of the system was very good. As future work, we would like to enhance the part of the application that generates the annotations, as this process could be done using Jape rules.

[illegible]

Furthermore, after deleting some of the generated rules, the

REFERENCES

- [1] C. M. Antunes and A. L. Oliveira, "Using context-free grammars to constrain apriori-based algorithms for mining temporal . . ." in *IN PROC. WORKSHOP ON TEMPORAL DATA MINING*, 2002.
- [2] R. Agrawal, H. Ho, F. Jacquenet, and M. Jacquenet, "Mining information extraction rules from datasheets without linguistic parsing," in *Innovations in Applied Artificial Intelligence, 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2005, Bari, Italy, June 22-24, 2005, Proceedings*, 2005, pp. 510–520.