UNIVERSITATEA “ALEXANDRU IOAN CUZA” IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Detectarea automată a efectelor adverse a medicamentelor**

propusă de

**Parasca Marius**

**Sesiunea:** Iulie, 2019

Coordonator științific

**Conf. Dr. Răschip Mădălina**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” IAȘI

**FACULTATEA DE INFORMATICĂ**

**Detectarea automată a efectelor adverse a medicamentelor**

**Parasca Marius**

**Sesiunea:** Iulie, 2019

Coordonator științific

**Conf. Dr. Răschip Mădălina**

Avizat,

Îndrumător Lucrare

Titlul, Numele şi prenumele \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Data \_\_\_\_\_\_\_\_\_\_\_\_ Semnătura \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DECLARAŢIE**

**privind originalitatea conţinutului lucrării de licenţă/disertaţie**

Subsemntatul(a) ………………………………………………………………………………………

domiciliul în …………………………………………………………………………………………………..

născut(ă) la data de ………………..…., identificat prin CNP ………….……………..………………..., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza” din Iaşi, Facultatea de ………………………. specializarea …………………………………………………………, promoţia ………………………….,

declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_elaborată sub îndrumarea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, pe care urmează să o susţin în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă/disertaţie să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării fasificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, ………………………… Semnătură student …………………………

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Detectarea automată a efectelor adverse a medicamentelor*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de testetc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Parasca Marius*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

**Cuprins**

[Motivație 6](#_Toc12858079)

[Introducere 7](#_Toc12858080)

[Capitolul I: Descriere generală 8](#_Toc12858081)

[Capitolul II: Preprocesarea datelor 10](#_Toc12858082)

[II.1 Tokenizare 10](#_Toc12858083)

[II.2 *Stemming* 11](#_Toc12858084)

[II.3 N-grame 12](#_Toc12858085)

[II.4 TF-IDF (Term frequency – Inverse document frequency) 15](#_Toc12858086)

[Capitolul III: Extragerea atributelor 18](#_Toc12858087)

[III.1 Conceptele si tipurile semantice UMLS 18](#_Toc12858088)

[III.1.1 UMLS (Unified Medical Language System) 19](#_Toc12858089)

[III.2 Expansiunea caracteristicilor folosind sinonime 21](#_Toc12858090)

[III.2.1 Aflarea părților de propoziție 22](#_Toc12858091)

[III.2.2 WordNet 22](#_Toc12858092)

[III.3 Alte caracteristici 24](#_Toc12858093)

[III.4 *SMOTE (Synthetic Minority Over-sampling Technique)* 26](#_Toc12858094)

[Capitolul IV: Metode de clasificare a datelor 28](#_Toc12858095)

[IV.1 Bayes Naiv 28](#_Toc12858096)

[IV.1.1 Model probabilistic 29](#_Toc12858097)

[IV.1.2 Exemplu 31](#_Toc12858098)

[IV.2 *SVM (Support Vector Machine)* 33](#_Toc12858099)

[IV.3 Rețele neuronale 37](#_Toc12858100)

[Capitolul V: Evaluare și rezultatele obținute 39](#_Toc12858101)

[Bibliografie 43](#_Toc12858102)

# Motivație

Detectarea precoce a efectelor adverse a medicamentelor după perioada de aprobare este o provocare crucială pentru tehnicile de farmacovigilență.

Farmacovigilența este definită ca: „activitatea și știința care se ocupă cu detectarea, înțelegerea și prevenirea efectelor adverse sau al oricărei alte probleme ale unui medicament”. Cercetătorii au arătat că efectele adverse cauzate de medicamente după ce au fost lansate pe piață este o problemă mare a sănătății publice. Prin urmare, supravegherea medicamentelor după punerea în circulație, este importantă atât pentru producătorii de medicamente, cât și pentru organizațiile internaționale ale sănătății.

În ultima vreme, interesul cercetătorilor din industria farmacologică a crescut semnificativ în zona de detectare automată a efectelor adverse a medicamentelor extrase din text din cauza creșterii foarte rapide a informațiilor legate de sănătate stocate in format electronic. Interesul a mai crescut și din cauza abilității computaționale pentru procesarea unor cantități mari de date automat, folosindu-se tehnici de procesare a limbajului natural și algoritmi de învățare automată.

# Introducere

Lucrarea de față este structurată pe 4 capitole care surprind elementele esențiale din rezolvarea problemei propuse.

**Capitolul I** cuprinde o scurtă descriere a pașilor urmați pentru rezolvarea problemei propuse. Pe lângă aceasta, tot aici se descrie corpusul folosit pentru antrenarea modelelor de clasificare.

**Capitolul II** cuprinde o descriere detaliată a fiecărui pas urmat în pregătirea datelor pentru a fi procesate de către clasificatori. În acest capitol sunt descrise concepte ca *stemming*, tokenizare, n-grame, *tf-idf*, etc. care sunt baza procesării limbajului natural.

**Capitolul III** cuprinde descrierea diferitelor utilitare folosite pentru selecția și extragerea atributelor (*MetaMap*, *WordNet*, etc.). Fiecare secțiune din acest capitol oferă și exemple referitoare la modul de funcționare a metodelor descrise de extragerea atributelor.

**Capitolul IV** cuprinde o descriere detaliată a fiecărei metode de clasificare (*Bayes* *Naive*, *SVM și rețele neuronale*) pentru problema care a fost abordată în lucrarea de față. Fiecare secțiune cuprinde o descriere detaliată a acestor metode de clasificare, împreună cu exemple ilustrative ale modului de funcționare.

**Capitolul V** descrie analiza experimentală a algoritmilor utilizați. Este prezentată evoluția acurateței fiecărui clasificator folosit după adăugarea a noi caracteristici. O analiză comparativă a celor doi algoritmi de clasificare folosiți pentru problema data este prezentată.

# Capitolul I: Descriere generală

Primul pas luat în realizarea lucrării de față este transformarea datelor dintr-un format ușor de înțeles de oameni într-un format ușor de înțeles de calculator, mai precis într-un format în care datele pot fi procesate ușor de modelele probabiliste și de algoritmii de învățare automată astfel încât clasificatorul format să fie unul care să facă preziceri bune.

Corpusul („baza de date”) folosit în lucrarea de față este unul public, și anume *ADE* corpus*[[1]](#footnote-1)* care conține instanțe (propoziții) care indică prezența sau absența unui efect advers al unui medicament, obținute din rapoartele unor cazuri medicale. Corpus-ul conține un total de 23516 de instanțe din care 6821 (aprox. 29%) conțin mențiuni a unui efect advers al unui medicament și 16695 (aprox. 71%) nu conțin mențiuni a unui efect advers al unui medicament.

După preprocesarea datelor, următorul pas este găsirea unui clasificator bun care să ofere o acuratețe mare pe setul de testare a datelor.

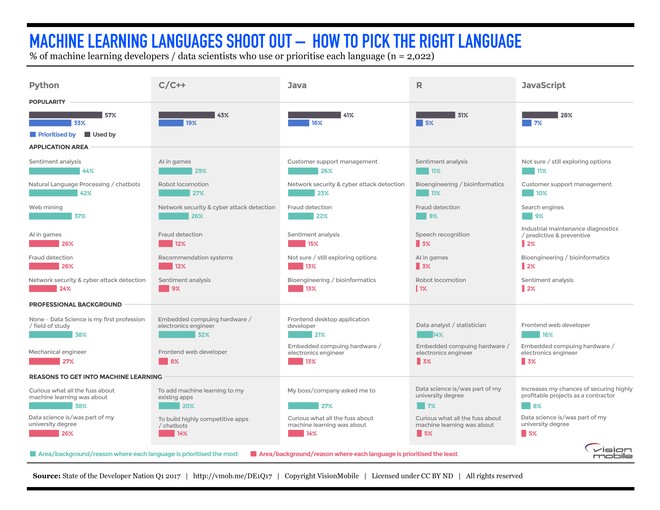
O problemă importantă reprezintă datele nebalansate. Pentru a obține o acuratețe bună a clasificatorului, este necesar ca numărul de instanțe pozitive să fie egal cu numărul de instanțe negative. Pentru acesta s-a folosit algoritmul SMOTE (*Synthetic Minority Over-sampling Technique*) de care vom vorbi mai pe larg in capitolul III.

Clasificatorii folosiți în aceasta lucrare sunt: *Bayes Naiv* si *SVM* (S*uport Vector Machine*) cu „*linear kernel”*. Acești clasificatori au oferit o acuratețe bună, *SVM*-ul oferind o acuratețe puțin mai ridicată decât *Bayes Naiv*. Acești clasificatori sunt descriși pe larg in capitolul III.

Ultimul pas este analiza experimentală a algoritmilor utilizati. A fost aplicat procedeul de *cross validation* pentru a testa că clasificatorul creat oferă rezultate bune în orice mod ar fi alese datele de testare și cele de antrenare astfel asigurându-ne că nu ne păcălim singuri și că, clasificatorul este într-adevăr unul bun.

Limbajul de programarea în care a fost scrisă lucrarea de față este *python*. Am ales *python* deoarece este un limbaj ușor de înțeles, cât mai apropiat de pseudocod și asta îl face să fie foarte lizibil și numai bun pentru procesarea limbajului natural deoarece se pot face prelucrări care conferă foarte puțin cod scris față de alte limbaje.

În *python* există multe biblioteci bine documentate care oferă API-uri (*Aplication Programming Interface*) cu care se pot face diferite preprocesări de text. De asemenea în *python* sunt scrise și foarte multe biblioteci care oferă algoritmi de clasificare cum ar fi *SVM, Bayes Naive*, etc.

Justificare alegerii limbajului de programare python este expusă in imaginea2 de mai jos.[[2]](#footnote-2)

În acestă imagine se poate observa că *python* devine din ce în ce mai popular în rezolvarea problemelor folosind algoritmi și tehnici de învățare automată.

# Capitolul II: Preprocesarea datelor

Acest capitol descrie fiecare pas urmat pentru a procesa datele dintr-un format ușor de citit de către oameni, într-un format ușor de citit și înțeles de către calculator. Acești pași de preprocesare sunt aplicați pe fiecare instanță din corpusul folosit astfel încât acesta va fi transformat într-un format în care datele pot fi procesate de către un algoritm de învățare automată.

## II.1 Tokenizare

Dându-se o secvență de caractere, tokenizarea este procesul de împărțire a secvenței în entități numite *token*-uri. Împărțirea se face pe baza spațiilor dintre cuvinte cât și anumitor caractere de punctuație.

*Token*-urile sunt de obicei cuvinte individuale, iar procesul de tokenizare este împărțirea în cuvinte individuale a unei secvențe de caractere. Acești tokeni sunt apoi folosiți ca intrări pentru diferite tipuri de algoritmi de preprocesare a limbajului natural.

**Exemplu:**

Intrare:

Friends, Romans, Countrymen, lend me your ears;

Output:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Friends | Romans | Countrymen | lend | me | your | ears |

[[3]](#footnote-3)

## II.2 *Stemming*

În lingvistica morfologică, *stemming*-ul este procesul de aducere a cuvintelor în formă inflexională (sau uneori a cuvintelor derivate) la forma de cuvinte de tip rădăcina din care s-au derivat sau la forma lor de bază.

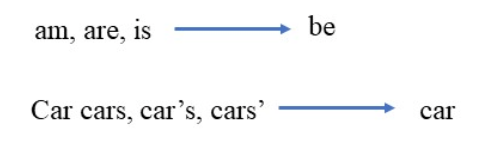
Algoritmii de *stemming* au fost studiați începând cu anii 1960 și până acum s-au dezvoltat diferiți algoritmi de *stemming* care diferă în funcție de performanță, acuratețe și de faptul a cum anumite obstacole au fost depășite.

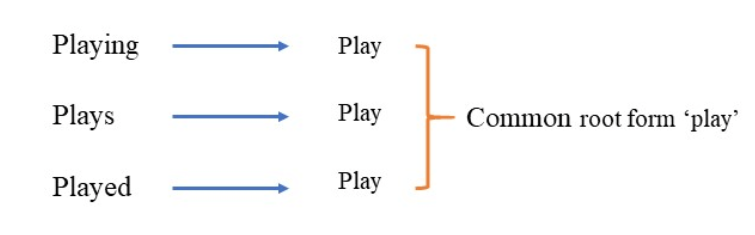
Algoritmii de *stemming* funcționează prin tăierea sfârșitului sau începutului unui cuvânt luând în considerare o listă de prefixe și sufixe care pot fi găsite într-un cuvânt în formă inflexională. Algoritmii de *stemming* pot produce două tipuri de erori: „*understemming*” si „*overstemming*”. „*Understemming*” apare atunci când două cuvinte ajung să aibă aceeași rădăcină, dar nu ar fi trebuit să aibă aceiași rădăcină – fals pozitiv. „*Overstemming*” apare atunci când doua cuvinte ar trebui să aibă aceeași rădăcină, dar nu au aceiași rădăcină – fals negativ.

Există multe timpuri de algoritmi de *stemming*, cum ar fi: *Potter’s stemming, Lovins stemmer, Dawson stemmer, Xerox stemmer*, etc. În această lucrare s-a folosit algoritmul *Potter’s stemming*.

Algoritmul Potter’s stemming este cel mai popular algoritm de stemming propus în 1980. Este bazat pe ideea că sufixele din limba engleză sunt formate dintr-o combinație de sufixe mai mici și mai simple. De exemplu dacă un cuvânt se termină in *EED*, se va înlocui *EED* cu *EE*: *agreed* se va transforma în *agree*.

**Exemplu:**



[[4]](#footnote-4)

Folosind asocierea de mai sus, următoarea propoziție va fi transformată după cum urmează:



## II.3 N-grame

În lingvistica computațională, o *n-*gramă este o secvență continuă de n elemente dintr-un text. Aceste elemente pot fi foneme, silabe, litere sau cuvinte în funcție de aplicație. N-gramele sunt de obicei extrase dintr-un text sau un corpus. Când elementele sunt cuvinte, n-gramele se mai numesc „*shingles*”.

Folosind prefixele numerice latine, o n-gramă de dimensiune 1 este numită unigramă, de dimensiune 2, bigramă, de dimensiune 3, trigramă, de dimensiune 4, 4-grama, și așa mai departe.

N-gramele sunt larg folosite în teoria probabilităților, teoria comunicării, lingvistică computațională (de exemplu: procesarea limbajului natural), biologie computațională (de exemplu: analiza secvențelor biologice) și compresarea datelor.

În aceasta lucrare am folosit n-gramele pentru procesarea limbajului natural, mai precis unigrame, bigrame și trigrame. Înaintea de a apela procedura de creare a n-gramelor, fiecare instanță din corpus va trece prin algoritmul de tokenizare si *steamming*. Pentru crearea unigramelor, bigramelor si trigramelor s-a folosit *CountVectorizer* din modulul *feature\_extraction* din libraria *sklearn*.

*CountVectorizer* functioneaza in felul următor: în primul rând se creează un vocabular din corpusul dat ca parametru. Vocabularul este un dicționar în care cheile sunt n-gramele, iar ca valori un număr de la 0 la numărul maxim de cuvinte (dacă se creează doar unigrame). Dacă se creează și unigrame și bigrame atunci numărul maxim va fi suma dintre numărul maxim de cuvinte si numărul maxim de bigrame. Valoarea numărului din dicționar reprezintă un index care va fi folosit ulterior.

Formula generală pentru dimensiunea maximă a dicționarului pentru 1 pana la n-grame:

După crearea vocabularului , se creează o matrice în care numărul de linii este numărul de instanțe din corpus, iar numărul de coloane este dat de formula de mai sus. Fiecare linie din matrice reprezintă o instanță din corpus în care fiecare index reprezintă o anumită n-grama (pentru 1-grama, index-ul reprezintă un cuvânt din vocabularul creat anterior), iar valoarea reprezintă de câte ori aceasta apare în instanța respectivă.

**Exemplu:**

Corpus:

|  |  |
| --- | --- |
| This is the first document | This is the second second document. |
| And the third one. | Is this the first document? |

Dicționarul pentru unigrame și bigrame va arăta în felul următor:

|  |  |
| --- | --- |
| Bigrama | Index |
| this is | 19 |
| is the | 6 |
| the first | 13 |
| first document | 4 |
| the second | 14 |
| second second | 11 |
| second document | 10 |
| and the | 11 |
| the third | 15 |
| third one | 17 |
| is this | 7 |
| this the | 20 |

|  |  |
| --- | --- |
| Unigrama | Index |
| this | 18 |
| is | 5 |
| the | 12 |
| first | 3 |
| document | 2 |
| second | 9 |
| and | 0 |
| third | 16 |
| one | 8 |

Iar tabelul care reprezintă de câte ori apare o unigramă sau o bigramă în corpusul dat ca intrare este următorul:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Counter | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

Această matrice va fi ca intrare pentru un algoritm de clasificare.

## II.4 TF-IDF (Term frequency – Inverse document frequency)

*Tf-idf* este o măsura populară in procesarea limbajului natural care precizează importanța unui termen dintr-un text. Termenii unici dintr-un segment de text vor primi un scor mai mare, iar cei comuni vor primi un scor mai mic.

Algoritmul *tf-idf* nu este decât o înmulțire a doi termeni al căror procedură de definire va fi descrisă mai jos:

1. Primul termen este acela de aflare a frecvenței fiecărui termen din document. În general documentele din care se extrage textul sunt mari. De exemplu, unu cuvânt care apare de 100 de ori într-un document nu face acel cuvânt de 100 de ori mai relevant pentru înțelegerea documentului respectiv, așa că folosim *log10* din frecvența unui termen. Prin urmare frecvența unui cuvânt se definește in felul următor:

Unde:

* *t* reprezintă termenul din document
* *d* reprezintă documentul în care termenul *t* se află
* *Count(t, d)* reprezintă de câte ori apare termenul *t* in documentul *d*

Prin urmare, un termen care apare de 10 ori în document va avea un tf=2, unul care apare de 100 de ori va avea un tf=3, unul care apare de 1000 de ori va avea un tf=4, și așa mai departe.

1. Al doilea termen este folosit pentru a da o pondere mai mare cuvintelor care apar doar în câteva documente. Termenii care aparțin doar unui număr mai mic de documente sunt folositori pentru diferențierea acelor documente față de restul colecției. Termenii care apar frecvent în întreaga colecție nu sunt atât de folositori. Frecvența termenului *t* al unui document, notat cu , nu este decât numărul de documente în care acesta există. În contrast , frecvența unui termen într-o colecție este numărul total de cuvinte care apar în orice document din întreaga colecție. De exemplu în colecția lui Shakespeare de 37 de piese de teatru[[5]](#footnote-5) , cuvintele *Romeo* și *action* au o frecvență de 113 (ambele apar de 113 ori în toate piesele de teatru), dar frecvența în documente este foarte diferită deoarece Romeo apare doar într-o singură piesă de teatru. Dacă scopul nostru este să găsim documente care conțin informații despre necazurile romantice ale lui Romeo, atunci cuvântul Romeo ar trebui să aibă o pondere mare.

|  |  |  |
| --- | --- | --- |
|  | Collection Frequency | Document Frequency |
| Romeo | 113 | 1 |
| action | 113 | 31 |

Atribuim o importanță mai mare cuvintelor caracteristice, cum ar fi *Romeo* prin frecvența inversă a documentelor, notat cu . -ul este definit de fracția , unde este numărul total de documente din colecție, iar numărul de documente în care *t* apare. Așadar cu cât termenul apare în mai puține documente, cu atât ponderea este mai mare. Ponderea mai mică de 1 este asignată termenilor care apar în toate documentele. Ce este de obicei un document? În operele lui Shakespeare, o piesă de teatru s-ar numi un document; dacă am procesa o colecție de articole despre enciclopedie cum ar fi articole de pe *Wikipedia*, atunci un document ar fi un singur articol. Se poate întâmplă ca corpusul să nu fie împărțit bine pe documente, atunci corpusul trebuie să fie divizat cumva în documente pentru a putea fi calculat .

Deoarece de obicei există un număr mare de documente într-o mulțime de colecții, vom comprima numerele folosindu-ne de funcția logaritm. Așadar rezultatul definiției pentru frecvența inversă a documentului () este:

Unde:

* *t* reprezintă termenul din document
* *N* reprezintă numărul total de documente din colecție
* reprezintă numărul de documente în care *t* apare

Mai jos este reprezentat un tabel care conține câteva cuvinte din corpusul lui Shakespeare, cuvinte care variază de la a fi foarte informative cum ar fi *Romeo*, la cuvinte care apar mai rar cum ar fi *salad* sau *Falstaff*, la cuvinte care sunt foarte comune cum ar fi *fool* sau altele care nu sunt deloc informative deoarece ele apar in toate cele 37 de piese de teatru: *good* sau *sweet*.

|  |  |  |
| --- | --- | --- |
| Word | df | idf |
| Romeo | 1 | 1.57 |
| salad | 2 | 1.27 |
| Falstaff | 4 | 0.967 |
| forest | 12 | 0.489 |
| battle | 21 | 0.074 |
| fool | 36 | 0.012 |
| good | 37 | 0 |
| sweet | 37 | 0 |

Așadar în final ponderea a unei valori pentru cuvântul *t* într-un document *d*, notat cu este definită ca:

Mai jos voi reprezenta doua tabele, primul va reprezenta numărul de cuvinte care apar în cele patru piese de teatru ale lui Shakespeare, iar al doilea tabel va conține exact aceleași cuvinte și piese de teatru doar că în loc de numărul de cuvinte din fiecare piesă de teatru, va conține valorile .

Tabel 1 (numărul de cuvinte care apar în cele patru de piese de teatru ale lui Shakespeare):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | As You Like It | Twelfth Night | Julius Caesar | Henry V |
| battle | 1 | 0 | 7 | 13 |
| good | 114 | 80 | 62 | 89 |
| fool | 36 | 58 | 1 | 4 |
| wit | 20 | 15 | 2 | 3 |

Tabel 2 (valorile pentru cele patru piese de teatru ale lui Shakespeare):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | As You Like It | Twelfth Night | Julius Caesar | Henry V |
| battle | 0.074 | 0 | 0.22 | 0.28 |
| good | 0 | 0 | 0 | 0 |
| fool | 0.019 | 0.021 | 0.0036 | 0.0083 |
| wit | 0.049 | 0.044 | 0.018 | 0.022 |

# Capitolul III: Extragerea atributelor

În acest capitol voi descrie tipul atributelor extrase din corpusul ADE, oferind și câte un exemplu a cum au fost extrase aceste atribute.

## III.1 Conceptele si tipurile semantice UMLS

Conceptele și tipurile semantice *UMLS* prezintă date medicale folositoare extrase din instanțele corpusului. Ca urmare le vom folosi ca și caracteristici (date de intrare) pentru algoritmul de clasificare. Așadar pentru acest tip de caracteristici vom calcula valorile *tf-idf* pentru conceptele și tipurile semantice extrase. *Tf-idf* este o măsură populară in procesarea limbajului natural și ne arată cât de important este un termen dintr-un text. Termenii care sunt unici într-un segment de text vor primi un scor mai mare, pe când termenii care sunt comuni în întregul corpus vor primi un scor mai mic.

### III.1.1 UMLS (Unified Medical Language System)

*UMLS*-ul este un compendium de vocabulare organizate din știința biomedicinei. Acesta permite translatarea dintre diferiți termeni din sistem; mai poate fi văzut ca o ontologie a conceptelor biomedicale. UMLS oferă și facilități pentru procesarea limbajului natural. Este în general creat pentru dezvoltatorii de sisteme informatice medicale.

*MetaMap* este o aplicație creată de *Lister Hill National Center for Biomedical Communications*  la *National Library of Medicine (NLM)* care realizează asocierea textelor medicale la *UMLS Metathesaurus* , sau pentru a identifica conceptele *Metathesaurus* aflate într-un text scris în limba engleză.

*Metamap* folosește o abordare bazată pe procesarea limbajului natural și a tehnicilor de lingvistică computațională, fiind folosit în lumea întreagă atât în industria academică, cât și în afara ei.

*Metamap* este o aplicație creată in Java care are funcționalități multiple, cum ar fi:

* extragerea informațiilor
* clasificare în funcție de anumite categorii
* extragere de concepte UMLS
* analizarea limbajului natural din literatura biomedicală si texte clinice
* înțelegerea textului
* data-mining

Din toate funcționalitățile oferite de *MetaMap*, în aceasta lucrare am folosit-o pe cea care oferă extragerea de tipuri semantice și a conceptelor din texte medicale. Voi expune mai jos fiecare pas efectuat pentru extragerea conceptelor si tipurilor semantice UMLS și transformarea lor în date pe care un algoritm de clasificare le poate procesa:

* extragerea conceptelor și tipurilor semantice folosind utilitarul *MetaMap* pentru fiecare instanță din corpus
* transformarea tipurilor semantice din abrevierile pe care *MetaMap* le returnează, la sensul lor de bază
* maparea fiecărui *id* al conceptului pe care *MetaMap* îl returnează, la translatarea pe care acesta o are
* concatenarea tipurilor semantice si a *id*-urilor conceptelor translatate
* aplicarea algoritmului pe datele identificate mai sus

**Exemplu pentru o singura instanță:**

Intrare:

|  |
| --- |
| Intravenous azithromycin-induced ototoxicity. |

**Pași urmați sunt următorii:**

|  |
| --- |
| ConceptMMI(index='tmplwdkzs1n',mm='MMI',score='19.24',preferred\_name='Azithromycin',  ***cui='C0052796',semtypes='[antb,orch]'*,**trigger='["Azythromycin"-tx-1-"azithromycin"-noun-0]',location='TX',pos\_info='13/12',tree\_codes='D02.540.576.500.992.050') |
| ConceptMMI(index='tmplwdkzs1n',mm='MMI',score='3.61',preferred\_name='Ototoxicity', ***cui='C0235280',semtypes='[inpo]'*,**trigger='["OTOTOXICITY"-tx-1-"ototoxicity"-noun-0]', location='TX', pos\_info='34/11', tree\_codes='') |
| ConceptMMI(index='tmplwdkzs1n',mm='MMI',score='3.47',preferred\_name='Induce(action)', ***cui='C0205263',semtypes='[ftcn]'*,**trigger='["Induced"-tx-1-"induced"-adj-0]',location='TX', pos\_info='26/7', tree\_codes='') |
| ConceptMMI(index='tmplwdkzs1n',mm='MMI',score='3.47',preferred\_name='Intravenous', ***cui='C0348016',semtypes='[spco]'*,**trigger='["Intravenous"-tx-1-"Intravenous"-adj-0]', location='TX', pos\_info='1/11', tree\_codes='') |

1. Extragerea obiectelor de tip Concept folosind *Metamap*:
2. Extragerea abrevierilor tipurilor semantice si *id*-urile conceptelor din lista de obiecte de tip Concept:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tipuri semantice | antb | ftcn | orch | inpo | spco |
|  |  |  |  |  |  |
| Id-uri Concepte | C0052796 | C0205263 | C0235280 | C0348016 | - |

1. Translatarea tipurilor semantice

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tipuri semantcie | Antibiotic | Functional Concept | Organic Chemical | Injury or Poisoning | Spatial Concept |

1. Maparea *id*-urilor conceptelor folosind un API REST[[6]](#footnote-6):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Concepte | Azithromycin | Induce (action) | Ototoxicity | Intravenous |

1. In ultimul pas se concatenează rezultatele și se aplica pe toate instanțele din corpus procesate în modul descris mai sus.

## III.2 Expansiunea caracteristicilor folosind sinonime

S-a arătat în cercetările din trecut că anumiți termeni dintr-o propoziție joacă un rol important (din cauza polarității lor anterioare) în determinarea polarității propoziției[[7]](#footnote-7).

Așadar anumite adjective, verbe și substantive și sinonimele lor sunt aproape invariabil asociate în determinarea polarității pozitive sau polarității non-pozitive. Prin urmare pentru fiecare adjectiv, substantiv și verb din propoziție am găsit termenii sinonimi folosind utilitarul *WordNet* și le-am folosit ca și caracteristici pentru algoritmul de clasificare. Am aplicat pe noua mulțime de date creată.

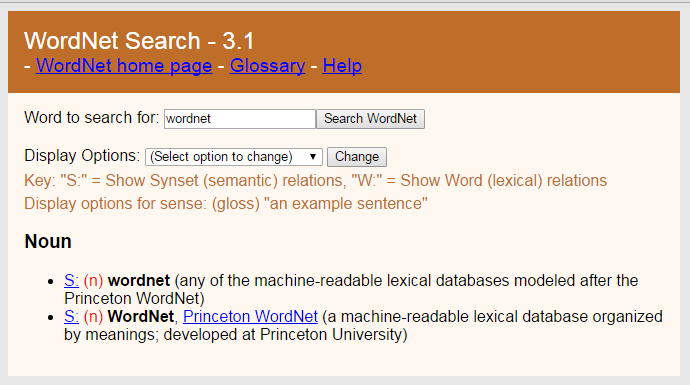
### III.2.1 Aflarea părților de propoziție

Pentru a găsi sinonimele adjectivelor, verbelor și substantivelor din propoziție, trebuie în primul rând să identificăm părțile de propoziție din fiecare instanță din corpus. Pentru aceasta ne-am folosit de algoritmul *pos\_tag* din biblioteca pentru procesarea limbajului natural din ptyhon: *nltk*.

Înainte de a apela algoritmul *post\_tag* pe o anumită instanță este nevoie ca instanța respectivă să fie preprocesată folosind un algoritm de tokenizare, cât și un algoritm de *stemming* (în cazul de față s-a folosit *Potter Stemer*) pentru a fi eligibilă pentru algoritmul de găsire a părților de vorbire[[8]](#footnote-8)

### III.2.2 WordNet

*WordNet* este un o bază de date lexicală pentru limba engleză. Grupează cuvintele în mulțimi de sinonime numite *synsets-uri*. Aceste *synset-uri* oferă o scurtă definiție a sinonimelor, cât și exemple în care acestea pot fi utilizate. Acestea sunt legate de alte sinonime sau membri printr-o structură de date. Prin urmare, *WordNet* poate fi văzut ca o combinație de dicționare, care este accesibil pentru utilizatorii umani prin intermediul web browser-ului, dar scopului lui principal este în analizarea automată a textului, cât și în aplicații de inteligență artificiala .

Interfața *WordNet* pentru utilizatori obișnuiți este prezentată în imaginea de mai jos:

Pentru dezvoltatori software, există atât baza de date cât și instrumentele software necesare sub licență *BSD*[[9]](#footnote-9) și pot fi descărcate gratis de pe site-ul *WordNet*[[10]](#footnote-10).

**Exemplu de expansiune a caracteristicilor folosind sinonimele oferite de *WordNet*:**

Intrare:

|  |
| --- |
| Pravastatin is associated with myotonia in animals. |

**Pașii urmați sunt următorii:**

1. Tokenization și *stemming*:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pravastatin | is | associ | with | myotonia | in | anim |

1. Aflarea părților de propoziție:

|  |  |
| --- | --- |
| Cuvant | Parte de propoziție |
| pravastatin | NN (substantiv) |
| is | VBZ (verb) |
| associ | JJ (adjectiv) |
| with | IN (prepozitie/conjucție coordonatoare) |
| myotonia | NN (substantiv) |
| in | IN (prepozitie/conjucție coordonatoare) |
| anim | NN (substantiv) |

1. Filtrarea rezultatelor:

|  |  |
| --- | --- |
| Cuvant | Parte de propoziție |
| pravastatin | NN (substantiv) |
| is | VBZ (verb) |
| associ | JJ (adjectiv) |
| myotonia | NN (substantiv) |
| anim | NN (substantiv) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PravacholSYN | pravastatinSYN | beSYN | personifySYN | constituteSYN |
| costSYN | make\_upSYN | embodySYN | existSYN | equalSYN |
| representSYN | followSYN | compriseSYN | liveSYN | myotoniaSYN |

1. Găsirea mulțimii de sinonime cu eticheta *SYN* atașată:
2. În ultimul pas se aplica pe întreg corpusul procesat în felul prezentat mai sus

## III.3 Alte caracteristici

Ultimele caracteristici create și concatenate cu celelalte caracteristici care formează datele de intrare pentru algoritmi de clasificare au fost următoarele:

* Lungimea instanțelor din corpus, măsurate în cuvinte
* Prezența adjectivelor comparative și superlative. Acestea sunt caracteristici binare, iar superlativele și comparativele sunt identificate folosind un *parser* care identifică părțile de vorbire
* Prezența modalelor: acestea sunt identificate folosind *parser*-ul care identifică părțile de vorbire

**Exemplu:**

**Intrare:**

|  |
| --- |
| 2-CdA induces lymphocytopenia, which may explain the improvement in this patient's psoriasis. |

1. Tokenizare, *stemming* și apoi concatenarea cuvintelor cu spațiu între ele:

|  |
| --- |
| 2cda induc lymphocytopenia which may explain the improv in thi patient psoriasi |

1. Aflarea părților de propoziție:

|  |  |
| --- | --- |
| Cuvânt | Parte de vorbire abreviată |
| 2cda | CD |
| induc | NN |
| lymphocytopenia | NN |
| which | WDT |
| may | MD |
| explain | VB |
| the | DT |
| improv | NN |
| in | IN |
| thi | JJ |
| patient | NN |
| psoriasi | NN |

1. Testarea prezenței superlativelor, comparativelor și modalelor și numărarea cuvintelor din instanță:

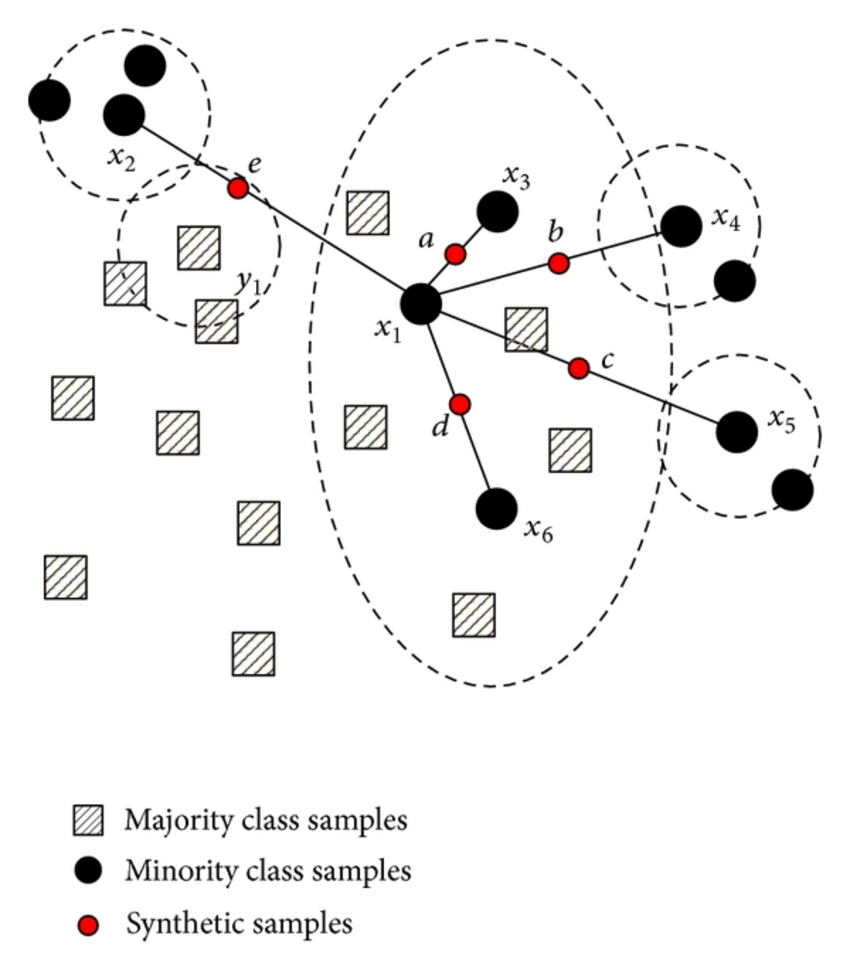
|  |  |  |  |
| --- | --- | --- | --- |
| Nr. cuvinte | Superlativ | Comparativ | Modal |
| 12 | 0 | 0 | 1 |

## III.4 *SMOTE (Synthetic Minority Over-sampling Technique)*

Există multe metode care fac *oversampling* datelor de antrenare pentru problemele tipice de clasificare. Una dintre cele mai cunoscute tehnici este *SMOTE (Synthetic Minority Over-sampling Technique)[[11]](#footnote-11),* acesta este un algoritm de balansare a datelor în cazul în care o anumită clasă a datelor de antrenare au o minoritate mare. De obicei clasificatoarele dau rezultate bune atunci când datele pentru clasificare au o rație 1 la 1.

Pentru a ilustra modul în care funcționează, considerăm că avem un set de antrenare cu *s* instanțe și *f* atributele. Pentru simplitate considerăm că aceste atribute au valori continue. Pentru exemplul de față considerăm o mulțime de date pentru clasificarea unor anumite tipuri de păsări. Spațiul caracteristicilor pentru clasa minoritară pentru care vrem să facem *oversample* ar putea avea următoarele atribute: lungimea ciocului, anvergura aripilor și greutatea (toate continue). Pentru a face *oversample*, considerăm o instanță din mulțimea de date și luăm în considerare toți cei k cei mai apropiați vecini (în spațiul de caracteristici al minorității). Pentru a crea o nouă instanță sintetică, luăm un vector al unui vecin, și punctul curent. Înmulțim vectorul cu un număr aleatoriu x între 0 și 1. Apoi adăugăm acesta la punctul nostru curent și așa am creat o nouă instanță sintetică.

Mai jos voi ilustra funcționalitatea algoritmului SMOTE. Considerăm că cercurile reprezintă o anumită specie de păsări (ea fiind de asemenea și cea minoritară), iar pătratele o alta specie de păsări:

[[12]](#footnote-12)

# Capitolul IV: Metode de clasificare a datelor

În acest capitol vom discuta metodele de clasificare utilizate. Având în vedere că avem o problemă de clasificare binară a datelor text, clasificatorul va clasifica pozitiv dacă simptomele pe care le prezintă pacientul au fost datorate consumului medicamentului precizat în text - efect advers al medicamentului respectiv. În caz contrar clasificatorul va clasifica instanța negativ.

Am utilizat următorii algoritmi de clasificare supervizată: *Bayes Naiv și SVM (Support Vector Machines).* Algoritmii de tip SVM sunt de obicei alegerea principală pentru problemele de clasificare pe text deoarece s-a observat că se descurcă destul de bine pentru acest tip de date din cauza capacității de a procesa spațiul de caracteristici cu o dimensiune foarte mare.

În cele ce urmează voi prezenta fiecare algoritm de clasificare în detaliu și voi furniza un exemplu pentru a înțelege cum funcționează fiecare.

## IV.1 Bayes Naiv

În învățarea automată, clasificatorii *Bayes Naives* reprezintă o familie de „clasificatori probabilistici” simpli, bazați pe aplicarea teoremei lui *Bayes* cu ipoteze puternice (naive) de independență între descriptori.

Clasificatorii *Bayes Naive* au fost studiați intensiv încă din anii 1960. A fost introdus (deși nu sub acest nume) în comunitatea recuperării de text la începutul anilor 1960 și a rămas o metodă populară pentru probleme de clasificarea a textului, pentru probleme de clasificare a documentelor ca aparținând unei anumite categorii (cum ar fi email din categoria spam sau legitim, sporturi sau politica, etc.) în funcție de frecvența cuvintelor din text ca și caracteristici (date de intrare) pentru algoritm. Cu o preprocesare a datelor bună, este un „adversar” competitiv în acest domeniu cu metode mai avansate ca *SVM.* De asemenea are aplicații și în diagnosticarea automată a pacienților.

Clasificatori *Bayes Naive* sunt scalabili, necesitând un număr de parametri liniar cu numărul de variabile (caracteristici) într-o problemă de învățare. Antrenarea cu probabilitatea maximă se poate efectua prin evaluarea unei expresii de formă închisă, care necesită timp liniar, față de aproximarea iterativă mai scumpă folosită pentru multe alte tipuri de clasificatori.

În literatura de specialitate de statistică și informatică, modelele *Bayes Naive* sunt cunoscute sub o multitudine de nume, cum ar fi *Bayes* simplu și *Bayes* independent. Toate aceste nume aduc referință la teorema lui *Bayes* ca regulă de decizie a clasificatorului, dar *Bayes Naiv* nu este neapărat o metoda bayesiană.

### IV.1.1 Model probabilistic

*Bayes Naiv* este un clasificator probabilistic. Considerăm că pentru un document *d*, clasificatorul va obține clasa din toate clasele . Această clasă având cea mai mare probabilitate maxim posterior a documentului respectiv. Realizarea acesteia se obține folosind următoarea formulă (1.1.1):

Unde:

* reprezintă estimarea clasificatorului pentru clasa corectă.
* reprezintă documentul
* reprezintă clasa
* reprezintă mulțimea tuturor claselor

Intuiția clasificatorului Bayesian este dată de folosirea formulei de transformare a lui *Bayes* pentru transformarea formulei (1.1.1) în alte probabilități care au proprietăți folositoare. Regula lui Bayes ajută la împărțirea unei probabilități condiționate în alte trei probabilități. Formula lui *Bayes* este descrisă mai jos (1.1.2):

Folosind formula lui *Bayes,* formula (1.1.1) se va transforma în (1.1.3):

Calcularea termenului pentru fiecare clasă posibilă duce la posibilitatea de eliminare a numitorului. Acesta se poate elimina deoarece calculăm probabilitatea fiecărei clase pentru documentul *d*, iar rămâne același de fiecare dată. Așadar putem simplifica formula (1.1.3) după cum urmează (1.1.4):

Prin urmare calculăm probabilitatea clasei pentru un anumit document *d* prin alegerea celui mai mare produs a două probabilități: probabilitatea a priori a clasei *c,* și *likelihood*-ul documentului *d*, . Pentru reprezentarea mai bună a formulei (1.1.4) o putem rescrie după cum urmează (1.1.5):

Fără a pierde din generalizare, putem reprezenta documentul *d* ca un set de atribute . Așadar formula se va transforma în (1.1.6):

Din nefericire, formula (1.1.6) este greu de calculat direct fără anumite simplificări. Estimarea probabilității fiecărei combinații posibile de atribute (de exemplu fiecare mulțime de cuvinte posibile și poziția acestora în propoziție) ar necesita un număr mare de parametrii și ar fi imposibil de antrenat pentru un set de antrenare foarte mare. Prin urmare, clasificatorul *Bayes* Naiv face doua presupuneri:

1. Prima presupunere spune că poziția unui cuvânt în propoziție nu contează. De exemplu cuvântul „*love”* are același efect pentru clasificator, chiar daca apare pe prima poziție sau pe poziția douăzeci din propoziție. Așadar presupunem că atributele codifică doar identitatea cuvântului nu și poziția acestuia în propoziție.
2. A doua presupunere este comun numită presupunerea *Bayes* Naiv. Acesta spune că probabilitățile sunt independent condiționale una de alta pentru o clasă *c* și prin urmare probabilitatea condițională poate fi împărțită în mai multe probabilități după cum urmeaz (1.1.7):

Deci din toate spune până acum formula finală de clasificare pentru *Bayes* Naiv este următoarea (1.1.8):

### IV.1.2 Exemplu[[13]](#footnote-13)

Problema clasificării persoanelor: clasificați dacă o persoana este bărbat sau femeie în funcție de anumite caracteristici măsurate. Caracteristicile includ înălțime, greutate și mărimea piciorului.

Exemplu de mulțime de antrenare:

|  |  |  |  |
| --- | --- | --- | --- |
| Persoană | Înălțime (m) | Greutate(kg) | Mărimea piciorului (cm) |
| bărbat | 1.85 | 82 | 30 |
| bărbat | 1.75 | 86 | 29 |
| bărbat | 1.73 | 79 | 32 |
| bărbat | 1.80 | 78 | 25 |
| femeie | 1.52 | 53 | 17 |
| femeie | 1.69 | 64 | 23 |
| femeie | 1.65 | 58 | 19 |
| femeie | 1.77 | 68 | 24 |

Clasificatorul creat din setul de antrenament folosind o distribuție ipotetic gaussiană ar fi:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Persoană | Medie (înălțime) | Varianța (înălțime) | Medie (greutate) | Varianța (greutate) | Medie (mărimea piciorului) | Varianța (mărimea piciorului) |
| bărbat | 1.7825 | 0.002168 | 81.25 | 9.68750 | 29 | 6.5 |
| femeie | 1.6575 | 0.008168 | 60.75 | 32.18750 | 20.75 | 8.1875 |

Să presupunem că avem clase cu probabilitate egală, astfel încât P(bărbat) = P(femeie) = 0.5. Această probabilitate a priori ar putea fi bazată pe cunoașterea de frecvențe pe o populație mia mare sau a frecvenței în setul de antrenament.

Mai jos avem un set de date pentru testare care trebuie clasificat:

|  |  |  |  |
| --- | --- | --- | --- |
| Persoană | Înălțime (m) | Greutate (kg) | Mărimea piciorului (cm) |
| test | 1.83 | 59 | 21 |

Dorim să determinăm care probabilitate posterioară este mai mare, bărbat sau femeie. Pentru clasificarea ca bărbat, probabilitatea este dată de:

Pentru clasificarea ca femeie, probabilitatea este dată de:

Media așteptată (numită, de asemenea, constantă de normalizare) poate fi calculată:

=

Cu toate acestea, având în vedere datele, media este constantă și, astfel, normalizează în mod egal ambele probabilități posterioare. Prin urmare, nu afectează clasificarea și poate fi ignorată. Acum putem determina distribuția de probabilitate pentru sexul datelor de test.

unde și sunt parametrii distribuției normale care au fost determinate anterior din setul de antrenament. Este important de reținut că o valoare mai mare decât 1 este în regulă aici – este o densitate de probabilitate în loc de o probabilitate, pentru că înălțimea este o variabilă continuă.

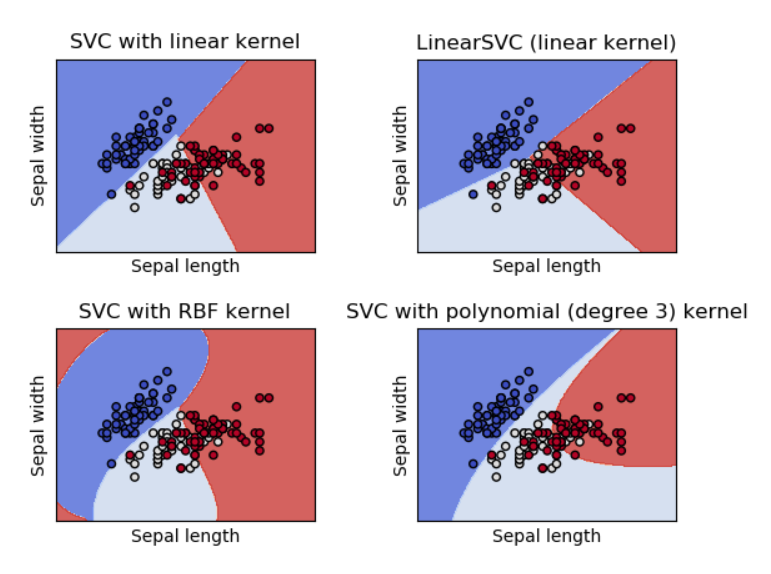
Deoarece numărătorul posteriorului este mai mare în cazul femeii, putem prezice că datele de test aparțin unei femei.

## IV.2 *SVM (Support Vector Machine)*

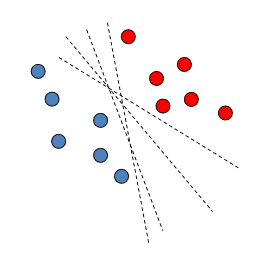
În învățarea automată, *support vector machines* sunt modele de învățare supervizată. Având o mulțime exemplu de antrenament, fiecare instanță fiind marcată ca aparținând uneia dintre cele două categorii, un algoritm SVM construiește un model care atribuie unor noi exemple una dintre cele două categorii, făcându-l astfel un clasificator binar liniar non-probabilistic (cu toate acestea exista SVM-uri care au o abordare de clasificare probabilistică, cum ar fi scalarea *Platt*). Un model SVM este reprezentat ca o mulțime de puncte în spațiu, mapate astfel încât categoriile sunt separate de o linie cu distanță dintre fiecare punct din fiecare categorie cu cât mai mare posibilă. Exemplele noi vor fi asociate unei categorii.

Mai formal, un *SVM* construiește un hiperplan sau o mulțime de hiperplane într-un spațiu dimensional mare sau infinit care poate fi folosit pentru clasificare, regresie sau în alte scopuri cum ar fi detectarea distorsiunilor. Intuitiv, o separare bună este atinsă atunci când hiperplanul are cea mai mare distanță față de cel mai apropiat punct de antrenare din oricare categorie (acesta distanță mai este numită și margine funcțională). În general cu cât este mai mare marginea, cu atât este mai mică eroare dată de clasificator.

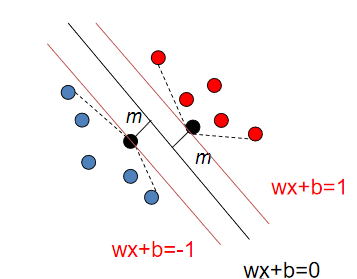
În timp ce problema originală poate să fi fost stabilită într-un spațiu dimensional finit, se întâmplă de obicei ca acele mulțimi din problema originală sa nu fie liniar separabile în spațiul stabilit. Spațiul original dimensional finit este transformat într-un spațiu dimensional mai mare, cu presupunerea că în noul spațiu separarea categoriilor va fi mai ușoară. Pentru a nu se folosi foarte multă putere computațională, asocierea dintre spațiile folosite de schemele *SVM* sunt proiectate pentru a ne asigura ca produsul dintre perechile de vectori de intrare pot fi calculați ușor în funcție de variabilele din spațiul original, prin definirea unei funcții numită „*kernel*” selectată pentru a se potrivi problemei.

Mai jos prezint câteva funcții *kernel* populare:[[14]](#footnote-14)

Considerăm o problema simplă de clasificare binară. Problema este liniar separabilă și se observa că există o infinitate de drepte (hiperplane) care permit separarea celor două clase. Ceea ce încearcă SVM să facă este găsirea unui astfel de hiperplan care să ducă la o generalizare cât mai bună nu doar pe datele de antrenare ci și pe datele de test.



Cel mai bun hiperplan este acela pentru care distanța minimă față de punctele aflate pe înfășurătoarea convexă a setului de puncte corespunzător fiecărei clase este maximă. Dreptele care trec prin punctele marginale sunt considerate canonice. Distanța dintre dreptele canonice este , deci a maximiza lărgimea zonei separatoare este echivalent cu a minimiza norma lui *w*.



Pentru a determina hiperplanul separator, se determină *w* și *b* care minimizează

(maximizează marginea separatoare) și satisface pentru toate elementele setului de antrenare , unde pentru clasa albastră și pentru clasa roșie.

Minimizarea lui astfel încât pentru , se poate rezolva folosind metoda multiplicatorilor lui Lagrange. Introducerea multiplicatorilor lui Lagrange transformă problema în determinarea punctului sa (*saddle point*) pentru V:

, unde este punct sa daca: .

Se construiește funcția duală:

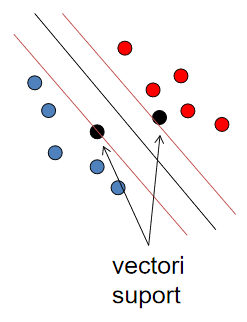
De aici se ajunge astfel la problema maximizării funcției duale în raport cu :

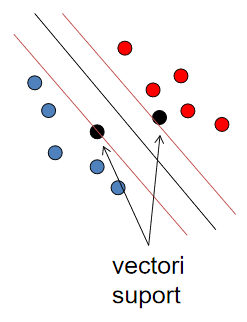
=, cu restricțiile: 0.

După rezolvarea problemei de mai sus (în raport cu multiplicatorii ) se calculează elementele hiperplanului separator astfel:

Unde *k* este indicele unui multiplicator nenul iar este exemplul corespunzător ce aparține clasei de etichetă +1.

Multiplicatorii nenuli corespund exemplelor pentru care restricțiile sunt active( *wx+b=1* sau *wx+b=-1*). Aceste exemple sunt denumite vectori suport și sunt singurele care influențează ecuația hiperplanului separator, celelalte exemple din setul de antrenare pot fi modificate fără a influența hiperplanul separator).

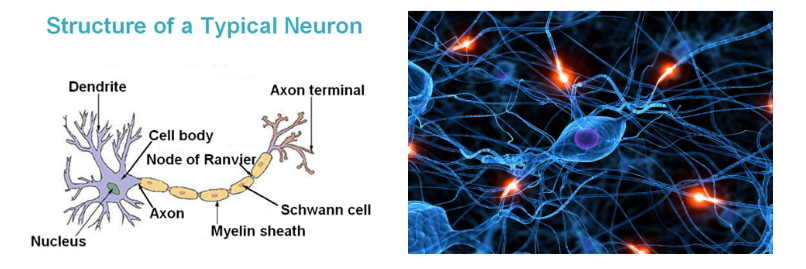


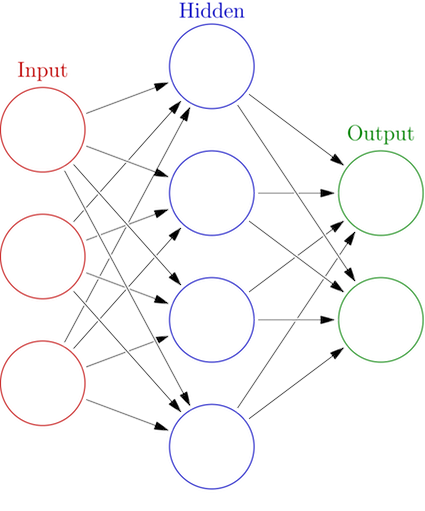


Multiplicatorii nuli corespund elementelor din setul de antrenare care nu influențează hiperplanul separator. Funcția de decizie obținută după rezolvarea problemei de optimizare pătratică este:

## IV.3 Rețele neuronale

Rețelele neuronale sunt clasificatori de tip *black-box*, permit predicția clasei dar nu furnizează reguli explicite de clasificare. Inspirația pentru rețele neuronale a venit inițial de la structura și funcționarea creierului (sistem de neuroni interconectați).





Rețelele neuronale artificiale sunt formate dintr-un set de neuroni artificiali (unități funcționale) interconectați. Fiecare neuron primește mai multe semnale de intrare și produce un semnal de ieșire. Neuroni sunt împărțiți în 3 mari categorii: neuronii de intrare (stratul de intrare), neuronii de ieșire (stratul de ieșire) și neuroni dintre stratul de ieșire și stratul intrare numiți si neuroni ascunși (stratul ascuns). Fiecare unitate (neuron) este simetric conectată cu toate celelalte unități din stratul următor.

Fiecare neuron este calculat ca suma ponderată a neuronilor din stratul precedent:

, unde este ponderea de pe arc, este valoarea neuronului, iar *b* este *bayas*-ul. După calcularea neuronilor, fiecărui dintre îi este aplicat o funcție de activare , unde *z* este valoarea neuronului după calcularea sumei ponderate.

Cele mai populare funcții de activare sunt:

* Sigmoid:
* Softmax:
* Relu:

O rețea neuronală este formată din trei mari algoritmi: algoritmul *Feed Forward* (calculează valoarea neuronilor folosind formula sumei ponderate), algoritmul *Gradient Descent* și algoritmul *Backpropagation*.

Algoritmul *Gradient Descent* adaptează ponderile și *bias*-urile pentru a minimiza funcția de cost. Funcția de cost este o funcție matematică care atribuie o valoare (un cost) care semnifică cât de rău cât de prost a fost clasificată o instanță.

Cele mai populare funcții de cost sunt:

* *Mean Square Error*:
* *Cross Entropy*:

Unde *w* reprezintă toate ponderile rețelei, *b* reprezintă toate *bias*-urile rețelei, *t* este vectorul de ieșire așteptat pentru vectorul de intrare *x*, iar *y* reprezintă ieșirea neuronului pentru inputul *x*.

Ultimul algoritm numit *Backpropagation* este și cel mai important algoritm, acesta având scopul de a propaga eroare costul înapoi la primi neuroni din stratul ascuns al rețelei neuronale.

# Capitolul V: Evaluare și rezultatele obținute

În acest capitol voi prezenta rezultatele obținute de algoritmii de clasificare pentru problema considerată. Tot aici voi expune și diferențele de acuratețe pe care clasificatorii le-au obținut în funcție de parametrii utilizați pentru antrenare.

În următorul tabel, voi prezenta evoluția acurateței pe baza adăugării de noi caracteristici extrase din corpus pentru clasificatorul *Bayes Naive*. Pe lângă adăugarea de noi caracteristici, un lucru care a îmbunătăți acuratețea pentru clasificatorul *Bayes Naive* a fost adăugarea unui algoritm de balansare a datelor (datele nefiind împărțite egal: 50% pozitive și 50% negative, ci procentul arată în felul următor: 29% pozitiv, iar 71% negativ) care este reprezentat în ultima coloană a tabelului.

Înainte de prezentarea rezultatelor voi prezenta pe scurt ce înseamnă acuratețea și scorul f1 al unui clasificator. Acuratețea este definită de următoarea formulă:

Unde:

* *TP* *– true positives*
* *TN – true negatives*
* *FP – false positives*
* *FN – false negatives*

Scorul f1 este de asemenea o metodă de măsurare a acurateței și este definită de următoare formulă:

Unde:

Mai jos voi prezenta rezultatele acurateței cât și rezultatele scorului f1 pentru primul experiment folosind clasificatorul *Bayes Naive*.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Bayes Naiv* | Caracteristici | | | |  |
| n-grame | *+ Tf-idf UMLS* | *+ Tf-idf* sinonime | + Alte  caracteristici | *+ SMOTE* |
| Acuratețe | 0.8469 | 0.8688 | 0.8742 | 0.8795 | 0.9103 |
| *F1-score* | 0.8508 | 0.8632 | 0.8738 | 0.8744 | 0.9054 |

În tabelul ce urmează voi prezenta aceeași evoluție ca în tabelul anterior, doar că de această dată folosind clasificatorul SVM cu *kernel* liniar.

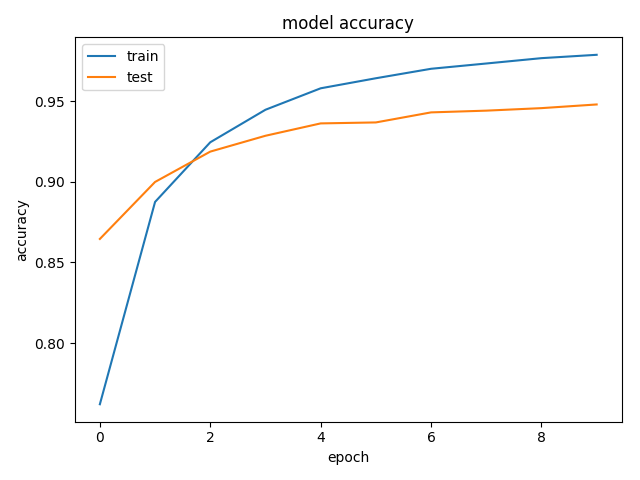
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *SVM*  *Linear*  *kernel* | Caracteristici | | | |  |
| n-grame | *+ Tf-idf UMLS* | *+ Tf-idf* sinonime | + Alte  caracteristici | *+ SMOTE* |
| Acuratețe | 0.9004 | 0.9088 | 0.9033 | 0.9016 | 0.9541 |
| *F1-score* | 0.8989 | 0.9076 | 0.9021 | 0.9000 | 0.9540 |

S-a încercat și o clasificare folosind rețele neuronale clasice cu următorul model:

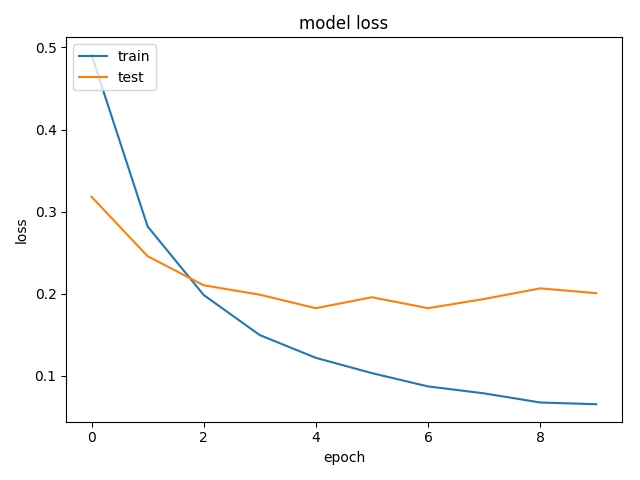
* *Input layer*
* *Dense layer(units=64, activation=’relu’)*
* *Dropout(.7)*
* *Dense layer(units=512, activation=’relu’)*
* *Dropout(.5)*
* *Dense layer(units=1, activation=sigmoid)*

*Loss function*-ul folosită este *binary\_crossentropy*, iar ca *optimizer* s-a folosit *adadelta*. Numărul de epoci rulate a fost de 10, cu un *batch size* de 70 datorită limitării *hardware.*

Graficul evoluție acurateței rețelei neuronale:



Iar graficul evoluției *loss function­*-ului se poate observa mai jos:



Rezultatele obținute în funcție de atributele folosite pe mulțimea de testare se poate observa în următorul tabel:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Neural Nets | Caracteristici | | | |  |
| n-grame | *+ Tf-idf UMLS* | *+ Tf-idf* sinonime | + Alte  caracteristici | *+ SMOTE* |
| Acuratețe | 0.8777 | 0.8842 | 0.8862 | 0.8544 | 0.9479 |
| Funcția de *loss* | 0.4717 | 0.4194 | 0.3924 | 0.3350 | 0.2008 |

# Bibliografie

[1] *Abeed Sarker, Graciela Gonzalez*

**Portable automatic text classification for adverse drug reaction detection via multi-corpus training**

*Journal of Biomedical Informatics Volume 53, Pages 196-207, February 2015* <https://www.sciencedirect.com/science/article/pii/S1532046414002317#b0230>

[2] *Daniel Jurafsky, James H. Martin*

**An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition; Third Edition draft**

*September 23, 2018,* <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>

[3] *Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze*

**Introduction to Information Retrieval, Cambridge University**

*Press. 2008,* [https://nlp.stanford.edu/IR-book/](https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html)

[4] *Steven Bird, Ewan Klein, and Edward Loper*

**Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit**, <https://www.nltk.org/book/>

[5] *Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer*

**SMOTE: Synthetic Minority Over-sampling Technique**

*Journal of Artificial Intelligence Research 16 (2002) 321–357,* <https://arxiv.org/pdf/1106.1813.pdf>

[6] *Andrew Ng*

**Part V, Support Vector Machines**

*CS229 Lecture notes*, <http://cs229.stanford.edu/notes/cs229-notes3.pdf>

[7] *Naiyang DengYingjie, TianChunhua Zhang*

**Support Vector MachinesOptimization Based Theory, Algorithms, and Extensions**

Chapman & Hall/CRC Data Mining and Knowledge Discovery Series,

<https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Support%20Vector%20Machines_%20Optimization%20Based%20Theory%2C%20Algorithms%2C%20and%20Extensions%20%5BDeng%2C%20Tian%20%26%20Zhang%202012-12-17%5D.pdf>

[8] **Stemming**, <https://en.wikipedia.org/wiki/Stemming>

[9] **Part of speech tagging**, <https://en.wikipedia.org/wiki/Part-of-speech_tagging>

[10] **Naive Bayes classifier**, <https://en.wikipedia.org/wiki/Naive_Bayes_classifier>

[11] **Support Vector Machine**, <https://en.wikipedia.org/wiki/Support-vector_machine>

[12] **Unified Medical Language System** <https://en.wikipedia.org/wiki/Unified_Medical_Language_System>

[13] *Indresh Bhattacharyya*

**SMOTE and ADASYN (Handling Imbalanced Data Set)**

*Aug 3, 2018,* <https://medium.com/coinmonks/smote-and-adasyn-handling-imbalanced-data-set-34f5223e167>

[14] **Oversampling and undersampling in data analysis**, <https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis>

[15] **SMOTE explained for noobs – Synthetic Minority Over-sampling line by line**, <http://rikunert.com/SMOTE_explained>

[16] **N-gram**, <https://en.wikipedia.org/wiki/N-gram>

[17] **WordNet**,<https://en.wikipedia.org/wiki/WordNet>

1. https://sites.google.com/site/adecorpus/home/document [↑](#footnote-ref-1)
2. https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7 [↑](#footnote-ref-2)
3. https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html [↑](#footnote-ref-3)
4. https://www.datacamp.com/community/tutorials/stemming-lemmatization-python [↑](#footnote-ref-4)
5. <http://www.1728.org/page10.htm>

   https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf [↑](#footnote-ref-5)
6. https://documentation.uts.nlm.nih.gov/rest/home.html [↑](#footnote-ref-6)
7. https://www.aclweb.org/anthology/I13-1084 [↑](#footnote-ref-7)
8. https://pythonprogramming.net/natural-language-toolkit-nltk-part-speech-tagging/ [↑](#footnote-ref-8)
9. https://en.wikipedia.org/wiki/BSD\_licenses [↑](#footnote-ref-9)
10. https://wordnet.princeton.edu/ [↑](#footnote-ref-10)
11. https://arxiv.org/pdf/1106.1813.pdf [↑](#footnote-ref-11)
12. https://medium.com/coinmonks/smote-and-adasyn-handling-imbalanced-data-set-34f5223e167 [↑](#footnote-ref-12)
13. https://en.wikipedia.org/wiki/Naive\_Bayes\_classifier [↑](#footnote-ref-13)
14. https://scikit-learn.org/stable/modules/svm.html [↑](#footnote-ref-14)