Cuprins

1. Introducere

Domeniul Inteligenței Artificiale este unul amplu care tratează probleme de actualitate oferind soluții eficiente în timp și performanță pentru secolul vitezei, în care de altfel trăim. Inteligența Artficială (denumită în continuare I.A), ramură a unui domeniu și mai complex - Informatica- nu ar fi existat fără calculatoare, acest lucru e clar. Pornind de la celebrul *test Turing*[1] și ajungând la sisteme expert evoluate care învață automat, sau mai mult decât atât, la *Internet-Of-Things (IoT)*[2], I.A. ne arată că principalul obiectiv este de a demonstra că mașinile pot deveni la fel de inteligente ca oamenii și de a putea folosi această abilitate în folosul omenirii.

1.1. Procesarea limbajului natural

Într-o societate care se dezvoltă pe zi ce trece din punct de vedere tehnologic, care trece peste granițele lingvistice și în care informația și comunicarea sunt elemente centrale ale oricărei activități, procesarea datelor oferă specialiștilor în lingvistică, științe socio-umaniste, tehnologia informației, calculatoare, inteligență artificială și nu numai, un temei solid pentru cercetarea domeniului: Procesarea Limbajului Natural.

Procesarea limbajului natural (eng. *Natural Language Processing – NLP și denumit în continuare astfel*) reprezintă o zonă de cercetare a I.A., care încearcă să rezolve aspectele înțelegerii și manipulării datelor de către un calculator. NLP se ocupă de varii probleme cu privire la limbajul natural, din care amintesc: căutarea și extragerea informațiilor, analiza și clasificarea conceptelor, subiectelor, sentimentelor și similarităților, ortografie, gramatică și stil, sumarizare și translatare, răspunderea la întrebări, sau recunoașterea vocii. Pentru căutarea și extragerea informațiilor, de exemplu, ajungem la un alt subdomeniu de studiu – *Data Mining*- care îmbină tehnicile de învățare cu statistici și sisteme de baze de date.

Totodată, și aplicațile de recunoaștere a vocii sunt o demonstrație vie a ceea ce poate face I.A. Celebrul profesor Stephen Hawking utilizează un astfel de sistem care sintetizează vorbirea, ajutându-l să comunice.

Pe de altă parte, de-a lungul timpului, s-a studiat intens și problema corectării gramaticale și ortografice a cuvintelor dintr-un text, a traducerii textelor sau sumarizarea acestora, evoluând de la soluții bazate pe reguli la cele automate, bazate pe învățare. În prezent, Google lucrează la un sistem ce permite traducerea în timp real a unui discurs, dar nu cuvânt cu cuvânt ci printr-o abordare ce se apropie de om, analizând și traducând bucăți din discuție.[3]

Procesarea limbajului natural constă și în cercetarea domeniului lingvistic pentru care se dorește realizarea sistemului, de multe ori soluțile nu permit altă tratare decât cea în care se pornește de la premisa că problema pe care dorim să o rezolvăm este dependentă de limbaj. Gândiți-vă la limba engleză, apoi la limba chineză.

[1] http://www.psych.utoronto.ca/users/reingold/courses/ai/turing.html

[2] http://blog.iquestgroup.com/en/iot-smart-cities/#.VWZGMfmqqko

3 http://www.descopera.ro/dnews/13761085-google-va-lansa-un-sistem-care-va-permite-traducerea-in-timp-real

1.2. Sisteme de procesare automată bazate pe *Machine Learning*

Sistemele de *Machine Learning*[4] (eng. ML și denumită în continuare astfel) învață automat din date. Această abordare reprezintă o alternativă la sistemul construit manual; în ultima decadă utilizarea M.L. a devenit o soluție apreciată și des folosită. Metodele de ML sunt deosebit de eficiente în situațile în care e necesară predicția pe seturi de date mari, diverse și variabile – *Big Data*[5]. În aceste seturi de date, ML depășește cu ușurintă metodele tradiționale pe precizie, scară și viteză.

Tehnicile de ML sunt utilizate în căutarea Web, plasarea anunțurilor, detecția fraudelor, sisteme de recomandare, recunoașterea amprentelor, fețelor și a vocilor, diagnosticarea medicală și multe altele.

O astfel de soluție am abordat și pentru problema despărțirii în silabe și a poziționării accentului cuvintelor din limba română, prezentată în continuare. Această problemă are o importanță aparte în diferite aplicații de NLP precum: procesarea „*text-to speech*", recunoașterea vocii și conversia literă-fonem.

Lucrarea de față este structurată după cum urmează. În capitolul 2 sunt descrise tema și obiectivele aplicației. Capitolul 3 prezintă studiul bibliografic efectuat trecând în revistă articolele și materialele folosite în elaborarea lucrării. Capitolul 4 cuprinde analiza sistemului prin justificarea soluției alese dar și detalierea componentelor utilizate. În următorul capitol, 5 se descrie elementele de proiectare, iar apoi detalile de implementare. Capitolul 6 cuprinde testarea și evaluarea aplicației, urmată de un scurt ghid de utilizare prezentat în capitolul 7. Capitolul 8 încheie prin prezentarea concluzilor.

[4] http://en.wikipedia.org/wiki/Machine_learning
[5] http://en.wikipedia.org/wiki/Big_data

2. Specificațile și Obiectivele Proiectului

2.1. Tema Proiectului

Silabificarea automată reprezintă procesul de separare a cuvintelor în silabe prin intermediul unui algoritm. Cunoștințele despre limitele silabelor afectează un număr de domenii de cercetare, deoarece baza lexicală este într-o continuă evoluție, prin apariția a noi cuvinte în vocabular. Zone de impact specifice ar putea fi reprezentate de prelucrarea automată *text-to-speech* (TTS), aplicații de detectare a limbii și recunoașterea vorbirii automată (ASR). O soluție de succes pentru problema silabificării automate ar fi de a dezvolta o soluție independentă de limbă, ușor adaptabilă la alte limbi și decuplate de utilizarea de dicționare.

Pe de altă parte, poziționarea accentului reprezintă problema marcajului literelor cuvântului cu un anumit accent. Limbile au diferite șabloane pentru accent. În timp ce în unele limbi, accentul este intuitiv și pentru nativii non-vorbitori, pentru altele locația accentului este complet imprevizibilă. Problema de predicție și de plasare a accentului a atras recent atenția oamenilor de știință din domeniul calculatoarelor, pentru că afectează aplicațile de TTS și cele de L2P- *letter-to-phonem*- dar și sistemele de recunoașterea vocii. În cererea de sisteme independente de limbaj, o soluție de succes mai bună și mai generală, ar fi identificarea de informații, cum ar fi amplasarea accentul silabic.

Proiectul de față presupune cercetarea, analiza și implementarea unui sistem de NLP, utilizând metodele de Machine Learning prin manipularea a două aspecte: despărțirea în silabe, respectiv poziționarea accentului cuvintelor din text, pentru limba română.

2.2. Obiectivele Proiectului

Așadar, scopul acestei activități de cercetare este realizarea unui sistem bazat pe învățare, utilizând metodele de *Machine Learning* care să rezolve problema despărțirii în silabe și a poziționării accentului pentru cuvintele din limba română. Unul dintre obiective este de a obține și o predicție de minimum 90% într-un timp cât mai scurt.

Un astfel de sistem care să prezică despărțirea în silabe și să identifice corect accentul cuvintelor, trebuie să aibă un model de date suficient de bun din care să învețe pentru a fi capabil să despartă în silabe sau să identifice accentul obținând un rezultat de cel puțin 90%. Modelul de date construit pentru învățare și trecut prin algoritmii de ML trebuie să conțină totodată un număr cât mai mic de cuvinte din care să învețe, dar predicția să rămână la rezultatul propus.

Totodată, despărțirea în silabe a cuvintelor dintr-un text în limba română presupune, în cea mai mare parte, respectarea regulilor impuse de gramatica limbii române. Spun în cea mai mare parte, deoarece există și excepții de la regulă ce necesită o abordare diferită. Una dintre acestea este ambiguitatea dată de structura diftong/hiat, exemplificată mai jos Tabel 1.1

Cealaltă problemă abordată, poziționarea accentului cuvintelor din text, presupune totodată cunoștințe despre silabele cuvântului. După cum se poate observa și din tabelul următor Tabel 1.1, numărul și formatul silabelor, alături de partea de vorbire au efect asupra poziției accentului dintr-un cuvânt.

În cazul poziționării accentului, trebuie menționat că în limba română accentul nu este definit explicit în scris, ci doar în vorbit, dar modelul de învățare trebuie să prezinte în construcție și accentul cuvintelor pentru a învăța corect (descris în detaliu ulterior).

Secvența	Cuvânt	Diftong	Cuvânt	Hiat
ai	Haină (subst.)	H**a**i-nă	Haină(adj.)	Ha-**i**-nă
oa	Soare	So**a**-re	Boar	B**o**-ar
ou	Cadou	C**a**-dou	Respectuos	Res-pec-tu-**o**s

Tabel 1.1. *Ambiguitatea diftong/hiat în gramatica limbii române pentru despărțirea în silabe și poziționarea accentului.*

Un alt obiectiv este de a testa sistemul pe un set de date considerabil de mare ca și dimensiune și de asemenea, de a crea modele de învățare bazându-se pe cuvinte despărțite corect și cu accentul bine poziționat. Acest aspect ne obligă să găsim date relevante pentru sistemul realizat.

Pe de altă parte, sistemul trebuie să fie scalabil, să permită predicția cuvintelor din texte de dimensiuni variabile și de ce nu, să permită și integrarea cu diferiți algoritmi de ML. Apoi, cum am menționat în rândurile de mai sus, sistemul trebuie să fie și performant din punct de vedere al timpului de răspuns.

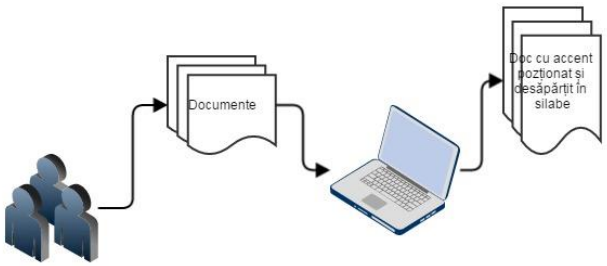


Figura 2.1. *Obiectivele aplicației*

2.3. Cerințe funcționale

Sistemul identifică accentul și realizează silabificarea textelor pe baza a patru tipuri de interacțiuni posibile cu utilizatorul. Astfel, acesta poate încărca fișiere text, selecta operația pe care o dorește – identificarea accentului sau despărțire în silabe-, selecta algoritmul de ML și vizualiza răspunsului sistemului.

Figura 2.2 descrie diagrama cazurilor de utilizare a sistemului. Cele mai importante cazuri sunt cele ce se referă la operații. Lucrarea de față descrie cu precădere cazul identificării accentului.

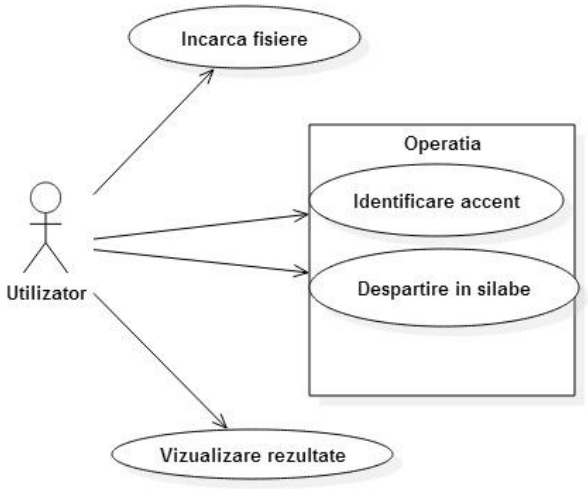


***Figura 2.2**. Diagrama cazurilor de utilizare*

2.4. Cerințe non-funcționale

Aspectele non-funcționale abordate se referă la scalabilitate, performanță, utilizabilitate, portabilitate, disponibilitate, extensibilitate.

2.4.1. Scalabiliate

Una din cele mai importante cerințe ce trebuie respectată este scalabiliatea. Principalele aspecte care pot oferi probleme de scalabilitate se referă la procesarea seturilor de date și apoi clasificarea acestora, necesitatea de a lucra cu volume mari fiind absolută. Aplicația trebuie să permită acomodarea mai multor resurse (mai multor tipuri de algoritmi, date ce vor fi procesate) și cerințe adiționale fără a schimba aplicația.

2.4.2. Performanță

De asemenea, aplicația trebuie să fie performantă din punct de vedere al timpului de răspuns și să aibă un timp minim de execuție în cazul generării fișierelor de ieșire. Totodată, putem să ne referim și la timpul de clasificare și construirea seturilor de date. Ceea ce e relevant pentru noi însă, este timpul de clasificare mai mult decât cel de antrenare sau de construire a seturilor

2.4.3. Utilizabilitate

Utilizabilitatea sistemului se referă la gradul de ușurință prin care se măsoară folosirea aplicației de către utilizator. Aplicația de față permite o utilizare fără probleme, prezentând o interfață simplă, precisă, ce nu lasă utilizatorul să se piardă. Aceştia au posibilitatea de a selecta operația pe care o doresc – despărțire în silabe sau identificarea accentului- să încarce fișiere sau să selecteze algoritmul.

2.4.4. Portabilitate

Aplicația trebuie să fie portabilă și să nu aibă eror Sistemul este implementat utilzând servicii Web, ca urmare aplicația trebuie să funcționeze indiferent de platformă, browser-ul folosit sau tehnologiile utilizate.

2.4.5. Disponibilitate

Aplicația trebuie să fie disponibilă 24/7 , dar cât timp rulează pe un sistem local, va fi disponibilă atât timp cât sistemul local rulează.

2.5. Soluția propusă

Soluția dezvoltată implică ambele sarcini, dar experimentele arată că este posibilă, de asemenea și tratarea independentă a problemelor. Abordarea presupune localizarea poziției accentului lexical după prezicerea silabelor pentru un anumit cuvânt. Avantajul acestei abordări este dată de acuratețea în marcarea silabelor și poziția accentului cu privire la timpul de procesare.

Soluția a fost dezvoltată și implementată împreună cu colega mea *Diana Maria BÂLC* de la specializarea Calculatoare linia engleză, drept urmare în continuare voi insista pe problema identificării accentului, partea de despărțire în silabe fiind detaliată în lucrarea „ *Supervised Learning For Romanian Syllabification Assignement*” a colegei mele.

2.5.1. Fluxul pentru identificarea accentului

Figura alăturată descrie procesul pentru identificarea accentului. Utilizatorul aplicației încarcă fișierul, sistemul îl parsează, verifică operația setată de utilizator, apelează procesul de identificare al accentului pentru ca în cele din urmă să întoarcă rezulatele.

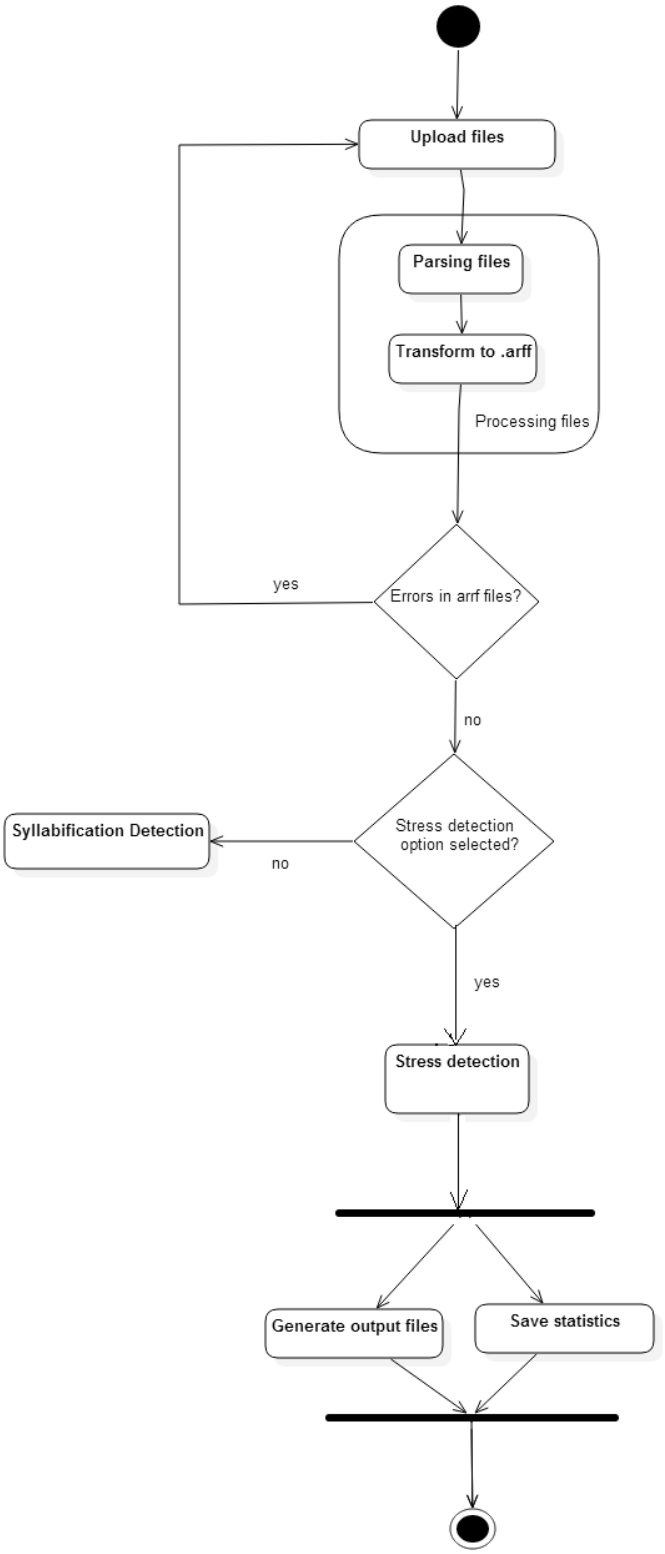


***Figura 2.3.**. Fluxul pentru identificarea accentului*

3. Studiu Bibliografic

Această secțiune descrie în detaliu cercetarea efectuată pentru probleme abordate. În prima, 3.1, vorbesc despre despărțirea în silabe, urmată de poziționarea accentului în secțiunea 3.2, pentru cuvintele din limba română. Explic, de asemenea și construcția setului de date necesar. În secțiunile 3.4 și 3.5 abordez învățarea bazată pe reguli, respectiv cea bazată pe antrenare.

Abordez și problema despărțirii în silabe deoarece cele două aspecte se influențează reciproc. Tabelul 3.1 arată câteva exemple care susține acest detaliu.

Caz	Cuvânt	POS[6]
Două variante de silabificare acceptate, aceași poziție a accentului	*i-gnór* *ig-nór*	Verb (present)
Parte de vorbire diferită, accent diferit	*e-chi-pá* *e-chí-pa*	Verb (trecut) Substantiv (singular)
Timp verbal diferit, accent diferit	*no-ti-fi-că* *no-tí-fi-că*	Verb (trecut) Verb (prezent)
Omografe[7]	*re-gíi* *ré-gii*	Substantiv (plural)
POS diferit, accent diferit, silabificare diferită	*bi-blio-gra-fi-á* *bi-bli-o-gra-fí-a*	Verb (prezent) Substantiv (singular)

Tabel 3.1. *Accent și silabificare. Influențare reciprocă.*

În cazul de față, prezicerea accentului se face după despărțirea în silabe pentru că informațile referitoare la silabe sunt utilizate în identificarea accentului. Acest aspect îl detaliez și analizez în capitolele următoare.

3.1. Despărțirea în silabe

În limba română există două moduri de despărțire în silabe: morfologic și fonetic. Apelez la un exemplu dat de despărțirea în silabe a cuvântului „*inegal",* pentru a evidenția diferența dintre cele două abordări.[8] Varianta morfologică impune silabisirea „*in-e-gal*" pe principiul că acest cuvânt este compus din prefixul de negație „*in*" și adjectivul „*egal*". Soluția fonetică, însă propune silabisirea „*i-ne-gal*".

Despărțirea în silabe are un rol esențial în limba scrisă pentru formatarea estetică a textelor și ușurința în citire. Nedespărțite în silabe, cuvintele lungi care nu încap la sfârșitul rândului, merg pe rândul următor lăsând un spațiu gol prea mare, spre exemplu.

În limba română, despărțirea în silabe se face după reguli clare, bine definite. Există însă și excepții de la regulă care sunt tratate separat.

[6] POS (eng. *Part-Of-Speech*) – partea de vorbire

[7] Omografe = cuvinte care se scriu la fel, dar se pronunță diferit-datorită accentului

[8] Cele două soluții pentru despărțirea în silabe sunt întâlnite și exemplificate în RoSyllabiDict, dicționarul utilizat pentru construcția seturilor de date. Despre mai multe informații despre dicționar, vezi secțiunea 3.5 Construcția setului de date

Conform *DOOM II*[9] și în funcție de ordinea gradului de acoperire, am realizat o clasificare a regulilor de despărțire în silabe prezentată în continuare. O dată cu acestea, descriu și excepțile de la regulă. Aproape toate regulile au și excepții.

3.1.1. Preliminarii

Fonologia limbii române [1] se ocupă cu studiul sunetelor din punct de vedere al valorii lor funcționale. Totodată, gestionează numărul de vocale și consoane –numite *unități segmentale*[10] - și determină accentul și intonația –acestea fiind numite *unități suprasegmentale*[11].

Literalele vocale în limba română sunt: *a, e, i, o, u, ă, â, î, w, y* ultimele două datorită neologismelor. Literalele semivocale sunt: *e, i ,o, u, ă, â, î, w, y*. Litera *a* poate fi doar vocală, nu și semivocală. Literalele consoane sunt celelalte litere rămase din alfabetul limbii române. Literalele vocale vor fi notate în continuare cu simbolul *V*, consoanele cu *C*, iar semivocalele cu *SV*. În continuare, fac o clasificare a regulilor de despărțire în silabe prezentând pentru fiecare caz șabloanele aplicate, exemple, respectiv excepțile de la regulă.

3.1.2. Reguli de despărțire în silabe bazate pe consoane

3.1.2.1. O consoană între două vocale trece în silaba următoare

Șabloane: V – **C**V, VS – **C**V, SVS – **C**V, V – **C**SV
Exemple: tá – **x**i, ié – **s**e, aú – **g**ust (subst.), lu – **p**oái – **c**ă
Excepție: O consoană urmată de *i* nu se desparte (*buni* adj., *flori, pomi, auzi, co – bori* vb.)

3.1.2.2. Două consoane între vocale se despart

Șabloane: V**C** – **C**V, VS**C** – **C**V, V**C** – **C**SV
Exemple: i**c** – **n**i, a – zvâ**r** – **l**i, a**c** – **t**iv, trai**s** – **t**ă
Excepții :

- Trec împreună la silaba următoare succesiunile de consoare care au ca al doilea element ***l*** sau ***r*** și ca prim element **b, c, d, g, h, p, t, v**, adică grupurile: **bl, br, cl, cr, dl, dr, fl, fr, gl, gr, hl, hr, pl, pr, tl, tr, vl, vr.**
- Nu se despart consoanele duble din cuvinte şi nume proprii cu grafii străine, care notează sunete distincte de cele notate prin consoana simplă corespunzătoare din limba română: ll [l'] (caudi-llo), zz [ţ]: (pi-zzicato), Negru-zzi

3.1.2.3. Trei consoane între vocale se despart după prima consoană

Șablon: **C** – **CC**
Exemple: o**b-şt**e, fi**l-tr**u, circu**m-sp**ect, deli**n-cv**ent, li**n-gv**ist, ci**n-st**e, co**n-tr**a, vâ**r-st**ă,
Excepții

- În următoarele succesiuni de trei consoane, despărţirea se face după primele două consoane:

[9] DOOM II- Dicționarul Ortografic, Ortoepic și Morfologic al Limbii Române, ediția a II-a
[10] Cf. DOOM II
[11] Cf. DOOM II

- **lp-t**: sculp-ta,
- **mp-t**: somp-tuos,
- **mp-ţ**: redemp-ţiune,
- **nc-ş**: linc-şii,
- **nc-t**: punc-ta,
- **nc-ţ**: punc-ţie,
- **nd-v**: sand-vici,
- **rc-t**: arc-tic,
- **rt-f**: jert-fă,
- **st-m**: ast-mul

- Se poate aplica despărţirea după structură în cazul cuvintelor:
  - compuse: alt|ceva, ast|fel, feld|mareşal, fiind|că, hand|bal
  - derivate cu prefixe: post – belic, trans – carpatic
  - derivate cu sufixe ca *-şor, -lâc, - nic*

3.1.2.4. Patru consoane între vocale se despart după prima consoană

Șablon: **C** – **CCC**
Exemple: a**b-str**act, co**n-str**uctor, î**n-zdr**ăveni

3.1.2.5. Cinci consoane între vocale se despart după a doua consoană

Șablon: **CC** – **CCC**
Exemple: ang|strom; opt|sprezece.

3.1.3. Reguli de despărțire în silabe bazate pe vocale

3.1.3.1. Două vocale alăturate se despart <u>sau</u> Vocalele în hiat se despart

Șablon: **V**-**V**(S), **V**-**V**C(C)
Exemple: **a**-**a**lenian; al**e-e**; l**e-u**l; f**i-i**nţă

3.1.3.2. Un diftong şi un triftong se despart de vocala sau de diftongul precedent

Șablon: (S)**V**-**S**V, (S)**V**-**S**VS, **V**-**S**SV
Când literalele e, i, o, u, w, z notează o semivocală, despărţirea se face înaintea lor.
Exemple: agr**e-eá**-ză, s**u-ie**, cr**e-ioá**-ne
Așadar, diftongii alăturați se despart: pl**oá-ie**, dum-n**ea-éi**

3.2. Poziționarea accentului

În limba română de regulă, accentul nu se notează grafic. Se păstrează totuși, accentul grafic din limba de origine, în neologisme – exemplu: *bourr**é**e-* și accentul din numele proprii străine – exemplu: *Moli**è**re* -, dar nu în toate : *Bogota, Panama, Peru* sunt doar câteva contraexemple ce pot duce la accentuări greșite.

Utilizarea explicită a accentului grafic este permisă în limba română pentru a evidenția cuvintele omografe dar neomofone care diferă sau nu prin poziția accentului: ***á**cele* (substantiv) - *ac**é**le* (adjectiv) , *copí**i** - c**ó**pii*, *v**é**selă* (adjectiv) - *ves**é**lă* (substantiv),

etc. Este permisă dar nu foarte utilizată în scrierea uzuală; din contextul textului, se determină sensul corect al cuvântului.

Poziția accentului într-un cuvânt are importanță și în silabificarea cuvântului. Există cuvinte în limba română care se scriu la fel, dar se citesc diferit datorită accentului. Acestea fac parte din grupul omografelor, exemplificat în tabelul 3.1.

Cuvânt	Silabificare	Cuvânt	Silabificare
H**a**ină (subst.)	H**a**i-nă	Ha**i**nă(adj.)	Ha-**i**-nă
Zăr**i** (subst.)	Zăr**i**	Z**ă**ri (verb)	Z**ă**-ri
R**ă**ți (subst.)	R**ă**-ți	Răț**i** (verb)	Ră-ț-**i**

***Tabel 3.2**. Despărțirea în silabe a omografelor. Poziția accentului în silabificare.*

De asemenea, pentru unele cuvinte mai vechi sau mai noi se admit și variante accentuate literare libere, astfel că un cuvânt poate avea două accentuări diferite[12] dar corecte : ac**a**tist / acat**i**st, ant**i**c / **a**ntic, m**a**nager / man**a**ger, etc.

De menționat că în limba română accentul se pune pe literele-vocale.

3.3. Tehnici bazate pe reguli

Primele metode folosite pentru despărțirea în silabe a cuvintelor au fost cele bazate pe reguli gramaticale[2]. Ideea presupune că pentru fiecare cuvânt se face o parcurgere a regulilor implementate în sistem, verificându-se astfel ce caz se potrivește.

Un astfel de sistem a implementat și Lorenzo Cioni pentru limba italiană și descris într-unul din articolele sale [3]. Autorul propune un algoritm recursiv, determinist, bazat pe arborii binari de decizie în care tratează problema despărțirii în silabe. Astfel, pentru fiecare cuvânt, algoritmul găsește o cale de la rădăcină și de la stânga la dreapta, identifică silaba, o salvează și se aplică recursiv pe restul cuvântului până la identificarea ultimei silabe, inclusiv. Totodată, se verifică și tipul literei- consoană sau vocală- pentru a găsi calea de parcurs. O frunză a arborelui reprezintă o regulă de despărțire în silabe. Pentru cuvântul „*mulțumesc*” cu despărțirea în silabe „*mul-țu-mesc*”, algoritmul se aplică recursiv pe toate cele trei silabe ale cuvântului conform figurii 3.1.

Un avantaj îl are faptul că în acest fel, excepțile sunt tratate la nivelul frunzelor- pentru un cuvânt ce nu e excepție, o frunză determină limita silabei. De menționat însă că în limba italiană accentul este poziționat explicit în scriere, iar în limba română nu. În același timp și regulile de despărțire în silabe diferă de la o limbă la alta. Algoritmul poate fi adaptat și utilizat pentru limba română, cazurile de excepție fiind tratate separat. Dintre acestea, o problemă aparte rămâne cea reprezentată de omografe – cuvinte care se scriu la fel dar se citesc diferit datorită poziției accentului - care se și despart în silabe diferit.

[12] Începând cu versiunea a doua a DOOM, se acceptă și două accentuări diferite a cuvintelor din fondul vechi sau nou al limbii române.

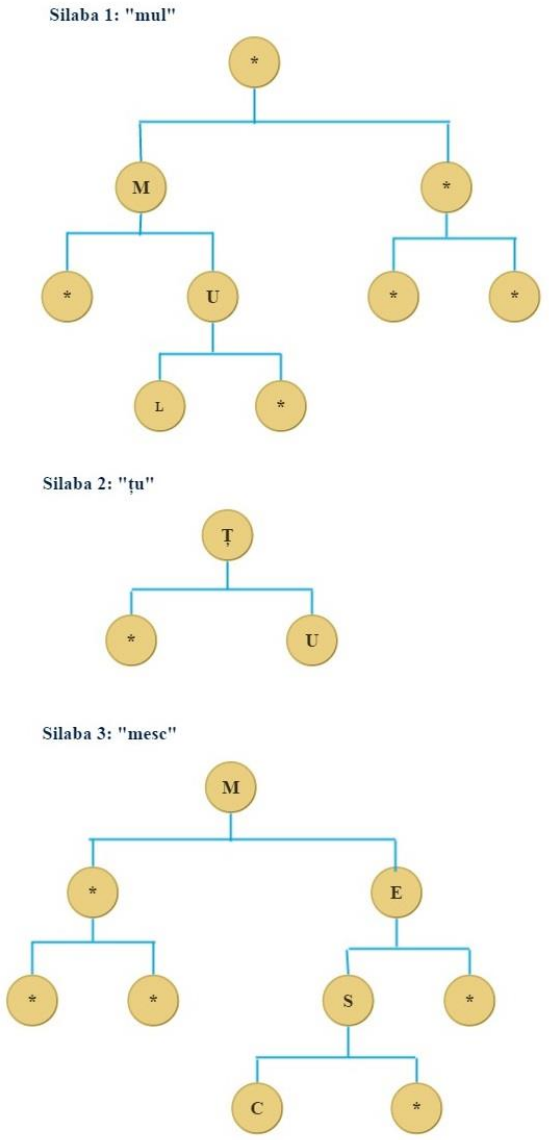


Figura 3.1. *Aplicarea algoritmului bazat pe arborele binar de decizie pentru despărțirea în silabe a cuvântului „mulțumesc*”.

3.4. Învățarea supervizată

În literatura de specialitate din ultimii ani, se vorbește des despre *Machine Learning* în NLP [5-7]. În primele metode folosite, cele bazate pe reguli, procesul nu se baza pe învățare, ci fiecare cuvânt era verificat cu regulile din gramatica limbii, iar excepțiile se tratau separat. Această nouă abordare, în schimb, se bazează pe antrenare – deducerea regulilor din exemplele date deja. Soluțiile întâlnite automatizează procesul de despărțire în silabe, respectiv de poziționare a accentului [5-8] utilizând algoritmi de *Machine Learning* pentru învățarea supervizată.

3.4.1. Preliminarii

În general, toți algoritmii de ML trebuie să fie antrenați pentru învățare supervizată –clasificare, predicție – sau învățare nesupervizată –clustering. Prin antrenare înțelegem o instruire/pregătire pe anumite intrări pentru ca mai târziu să le putem testa pe intrări necunoscute, ne mai întâlnite pe care să le clasificăm sau anticipa. Pe aceasta se bazează cei mai mulți algoritmi de învățare, cum ar fi : SVM, Rețele Neuronale, Naive Bayes, etc.

Într-un proiect de ML, seturile de date trebuie împărțite, având: un set de dezvoltare – set de antrenare + set de test – și un set de test/ evaluare. Setul de date de test trebuie să aibă același format ca și cel de antrenare, dar e important ca cele două să fie diferite din punct de vedere al corpus-ului: dacă pur și simplu reutilizăm seturile de date de antrenare și de test, modelul va memora intrarea fără a învăța cum să generalizeze noi exemple și ar primi în mod eronat rezultate foarte bune.

1. Procesul de învățare supervizată

În procesul de învățare supervizată, avem un set de date constând în trăsături (*eng- features*) și etichete (*eng-labels*). Sarcina este de a construi un estimator care capabil să prezică eticheta unui obiect dat de setul de caracteristici. Clasificarea este sarcina de a prezice valoarea unei variabile (instanțe) dat de alte variabile de intrare (caracteristici sau "predictori").

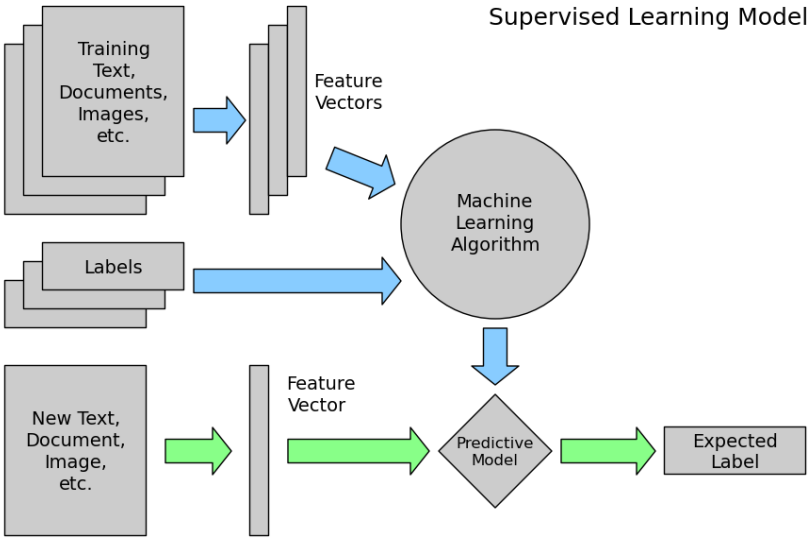


Figura 3.2. Procesul de învățare supervizată[13]

Figura 3.2 schițează modelul de învățare supervizată. Pornind de la diferite documente, texte, imagini se extrag trăsăturile pe baza cărora se construiește vectorul de trăsături (*eng – feature vector*). Algoritmii de ML sunt aplicați pe noile seturi obținând un model predictiv, antrenat. Acest model este utilizat în continuare pe alte seturi de date de

[13] http://www.astroml.org/sklearn_tutorial/_images/plot_ML_flow_chart_1.png

testare-parsate pentru a se putea extrage trăsăturile și crea vectorul de trăsături- efectuând astfel clasificarea.

Vectorul de trăsături, după cum îi spune și numele, este un vector cu toate caracteristicile extrase adăugând în plus pe ultima poziție și clasa de predicție. Se construiește câte un vector pentru fiecare instanță din setul de date. Acesta are o formă generală, clară pe care toate instanțele trebuie să o respecte. Figura 3.3 prezintă funcționalitatea de bază din procesul de învățare supervizată.

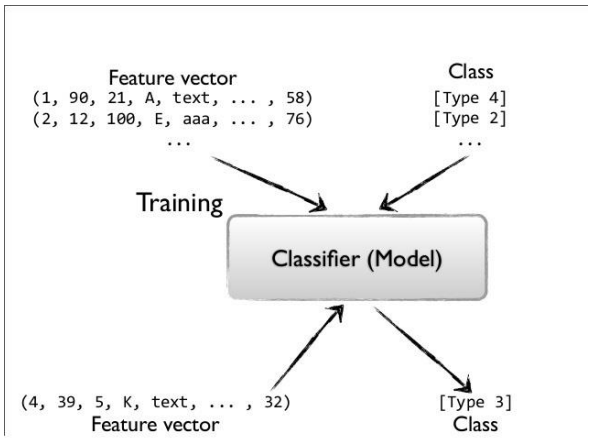
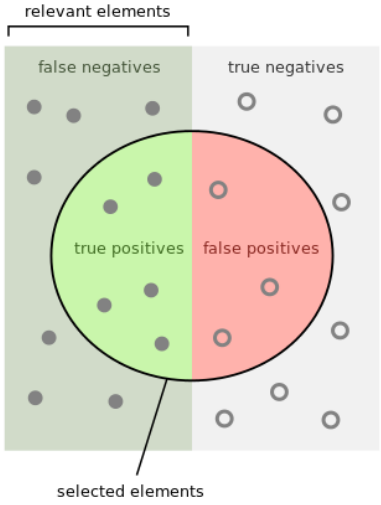


Figura 3.3. Core-ul procesului de învățare supervizată

2. Metrica procesului de învățare supervizată

Atunci când evaluăm clasificarea efectuată, analizăm următoarele metrici:

- FP Rate = rata false-positive - instanțe clasificate incorect pentru clasa dată (eronat, instanța a fost detectată).
- TP Rate = rata true-positive - instanțe clasificate corect pentru clasa dată.
- FN Rate = rata false-negative – instanțe neclasificate dar care ar fi trebuit clasificate
- TN Rate = rata true-negative –instanțe neclasificate, care trebuie să fie neclasificate



$$Precision: \frac{TP}{TP+FP}$$

$$Accuracy: \frac{\sum TP + \sum TN}{\sum INSTANCES}$$

$$Recall: \frac{TP}{TP+FN}$$

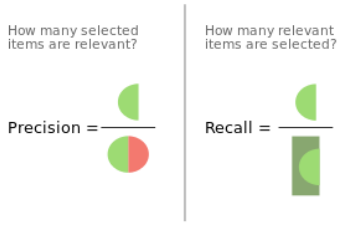


Figura 3.4. Core-ul procesului de învățare supervizată[14]

Mean : $\frac{1}{n}\sum x_i$ unde n – nr total de instanțe, i = {1,n}

Standard Deviation : $\sqrt{\frac{1}{n-1}\sum(x - \text{mean})^2}$

Precision și *Recall* sunt invers proporționale. Pe măsură ce una crește, cealaltă scade.

3.4.2. Algoritmi de învățare supervizată

Principalul obiectiv al tehnicilor de învățare supervizată este de a clasifica un număr mare de instanțe în clase. Procesul de învățare are loc în doi pași. Antrenarea crează un model utilizat pentru învățare, iar predicția utilizează modelul pentru a învăța, a stabili eticheta noii instanțe. Practic, clasificarea atașează o clasă/etichetă cu un anumit set de caracteristici unei noi instanțe pe baza etichetei cunoscute dinainte (învățate din modelul antrenat). Figura 3.2 descrie schematic pașii în procesul de învățare.

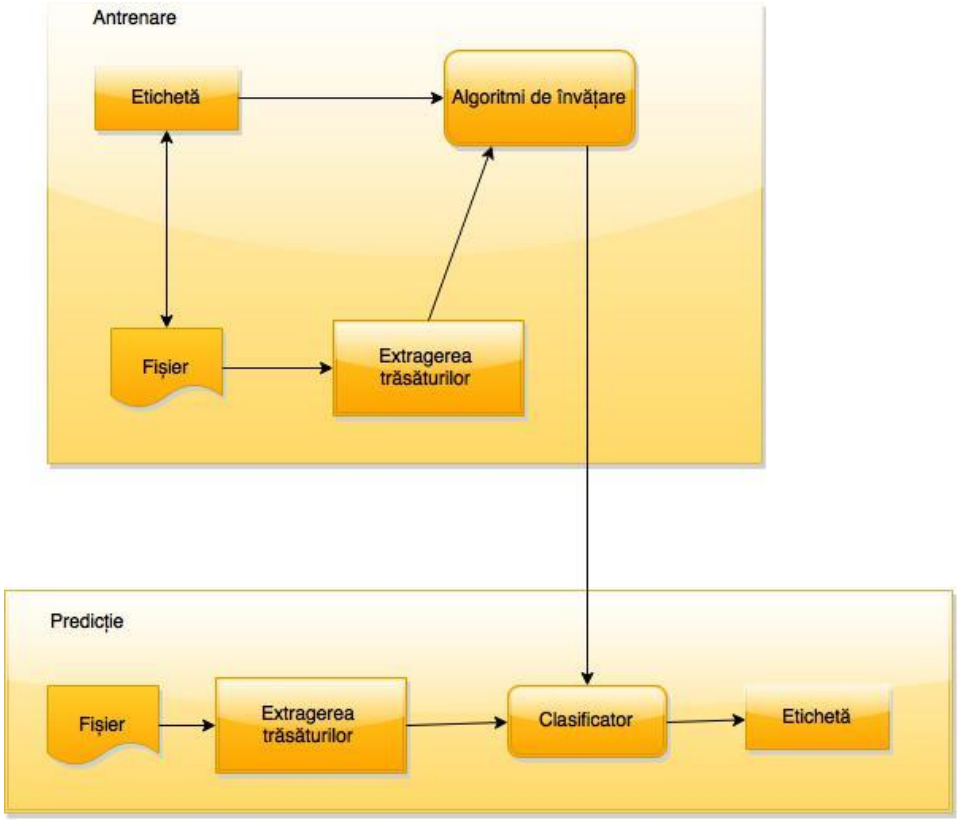


Figura 3.2. Procesul de învățare supervizată

[14] https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg

3.4.2.1. Support Vector Machine - SVM

Unul dintre cei mai utilizați algoritmi de clasificare în procesarea limbajului natural este SVM-*Support Vector Machine*. Algoritmii de SVM sunt capabili să analizeze datele și să recunoască modelele folosind metode de clasificare liniar non-probalistice. Instanțele din setul de antrenare care aparțin claselor sunt reprezentate ca puncte în spațiu. Se bazează pe planurile de decizie și definește limitele de decizie, separând setul de obiecte cu clase diferite de un decalaj. Cu cât decalajul dintre reprezentările spațiale ale instanțelor e mai mare, cu atât clasificarea unei noi instanțe este mai ușoară. Mai mult decât atât, implică și găsirea unei funcții corespunzătoare care să partiționeze instanțele în acest spațiu în concordanță cu clasa prezisă și care să găsească cea mai bună linie de separare (Figura 3.3)

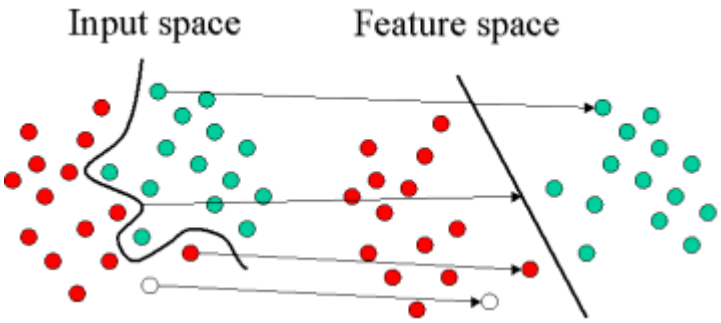


Figura 3.3 [15]*Clasificarea instanțelor de către algoritmul SVM*

3.4.2.2. Random Forest - RF

RF este un algoritm de Machine Learning care lucrează cu arborii de decizie pentru orice probleme de clasificare sau regresie. Algoritmul este robust pentru diferite tipuri de variabile de intrare sau date lipsă și s-a demonstrat că este performant cu seturi mari de date.[16] Acesta generează arbori de decizie în mod aleatoriu, specific, astfel încât fiecare e necorelat cu ceilalți. Mai exact, arborii sunt antrenați pe același set de date pentru a fi necorelate, prin utilizarea subseturilor de trăsături eșantionate aleatoriu pentru evaluarea la fiecare nod al fiecărui arbore și un subset de puncte de date (eșantionate tot aleatoriu) pentru antrenarea fiecărui arbore.

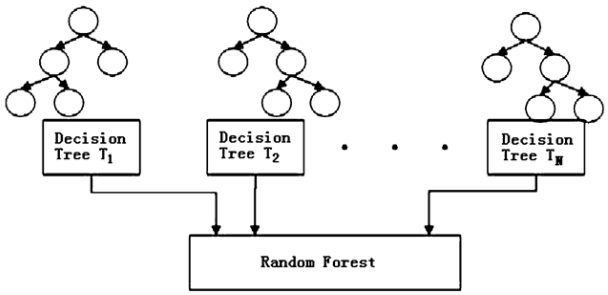


Figura 3.4 *Random Forest*[17]

[15] http://www.statsoft.com/textbook/graphics/SVMIntro3.gif
[16] http://www.kenvanharen.com/2012/02/whats-best-machine-learning-algorithm.html
[17] http://what-when-how.com/wp-content/uploads/2012/06/tmp35b0459_thumb.png

Partea bună a arborilor este că au flexibilitate în ceea ce privește tipul de date de intrare și ieșire care pot fi o valoare categorică, binară și numerică. Nivelul de noduri de decizie indică, de asemenea gradul de influențe ale diferitelor variabile de intrare.. De asemenea, criteriile de decizie ia în considerare o singură intrare atribuite la un moment dat, nu o combinație de mai multe variabile de intrare. Un punct slab al arborilor este că o dată ce a fost învățat nu poate fi actualizat incremental. Când sosesc date noi de antrenare, se va reface arborele de la zero.

3.4.2.3. Naive Bayes

Algoritmul poate fi tratat ca și un graf de dependențe în care fiecare nod reprezintă o variabilă binară și fiecare muchie (direcțională) o relație de dependență. Dacă nodul X1 și X2 au o dependență cu nodul C (figura 3.5), probabilitatea ca C să fie adevărat depinde de diferite combinații ale valorilor boolene ale nodurilor X1 și X2. Teorema lui Bayes oferă o formulă de calcul.

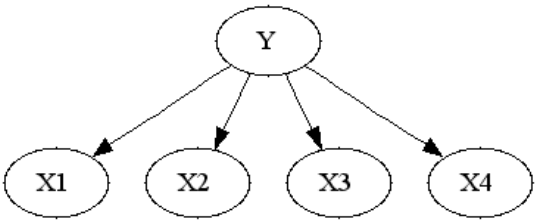


Figura 3.5 *Naive Bayes*[18]

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

3.4.2.4. Ada Boost

Boosting este o altă abordare, bazată pe ideea de a crea o regulă de predicție precisă prin combinarea mai multor reguli inexacte. În general, este utilizat în combinație cu alți algoritmi de învățare, mai slabi obținând soluții mai eficiente. Ce atrage atenția este că procesul de antrenare selectează doar acele trăsături care cresc puterea de predicție a modelului, reducînd dimensiunea și îmbunătățind timpul de execuție (comparativ cu SVM la care timpul de execuție e mult mai mare).

Clasificatorii slabi sunt aceia puțini corelați cu o clasificare adevărată. Aceștia sunt clasificatori care au un indiciu cu privire la modul de prezicere a etichetelor corecte, dar nu la fel de multe ca și clasificatorii puternici precum: Naive Bayes, Rețele Neuronale sau SVM. Unul dintre cei mai simplii clasificatori slabi este Stump Decision. Acesta selectează un prag pentru o caracteristică și împarte datele de pe acest prag.

*3.4.2.5. KStar – K**

KStar[9] este un clasificator bazat pe instanță a cărui clasă pentru o instanță de test este determinată de instanțele de antrenare similare. Funcția utilizată este una de distanță bazată pe entropie.

3.4.2.6. Conditional Random Fields – CRF

CRF[10] la rândul său se utilizează des în *Machine Learning*, în special în cazurile în care se dorește o predicție structurată[2,14]. Întrucât un clasificator obișnuit prezice o etichetă pentru o singură instanță, indiferent de vecini, CRF poate lua în

[18] http://yaroslavvb.com/research/reports/naive_bayes/graph1.png

considerare și contextul prezicând secvențe de etichete pentru secvențe de eșantioane de intrare - ceea ce îl face popular în procesarea limbajului natural.

3.4.3. Soluții pentru învățarea supervizată a identificării accentului și silabificării

Pentru problema identifiicării accentului și a silabificării, în literatura de specialitate găsim diferite soluții de implementare în funcție de limbă, trăsături și performanță.

Liviu Dinu utilizează SVM în predicția accentului[10][11][12] și a despărțirii în silabe[5]. Mai mult decât atât, face o analiză a clasificatorului comparând rezultatele cu CRF și un sistem bazat pe reguli – pentru despărțirea în silabe. SVM aduce o performanță mult mai bună decât celelalte abordări.

Tot cu SVM propune și E. Oancea rezolvarea problemei legate de identificarea accentului [13]. Acesta vine cu un algoritm ce ia în considerare caracteristicile morfologice, fonetice și lexicale ale cuvântului.

Pentru limba română, un sistem care învață din antrenarea datelor cunoscute deja este cel descris de Tiberiu Boroș [6]. MIRA (*eng. Margin Infused Relaxed Algorithm*) căci despre el este vorba, vine cu o soluție atât pentru poziționarea accentului, cât și pentru despărțirea în silabe sau determinarea formei canonice a cuvântului. Sistemul este relativ nou (2013) și folosește *clasificatorul Perceptron* pentru a învăța. Din cele descrise de autor, rezultatele sunt mult mai bune comparativ cu sistemele ce utilizează alți clasificatori.

Un alt articol ne vorbește despre un sistem bazat pe învățare folosind *Structured – SVM* [14]. Soluția pare la fel de fiabilă ca cea bazată pe SVM [5], descrisă de L.Dinu și mai de încredere decât soluția lui T.Boroș descrisă anterior, datorită popularității în utilizare. Structured-SVM permite antrenarea etichetelor de ieșire, structurate, comparativ cu SVM care permite clasificarea binară sau multiclasă, dar nu și cea structurată; acest aspect poate fi un avantaj: învățarea structurilor, nu învățarea nominală.Ca și o problemă de clasificare structurată este luată în considerare de către Barlett [14] și Kondrak [15]. Autorii cred că există modele, atât pentru identificarea accentului cât și pentru silabisire datorită regularităților observate prin exemple corecte. În articolele[14,15] aceștia menționează de *Hidden-Markov-Models* și SVM (SVM-HMM) ca și metode utilizate. Mai mult decât atât, ei arată că formularea problemei are nevoie de o schemă de etichetare, o modalitate de a anota datele - etichetarea pozițională (*Not Boundary, Boundary – NB tags*) și etichetarea structurală (*Onset, Nucleus și Coda*). Pentru silabificare de exemplu, fiecare silabă este compusă dintr-o secvență de foneme: nucleus (vocală) precedată de un onset (consoană) și urmată de o coda (consoană). Din punct de vedere al fonemului, nucleus și coda dau rima.

O altă abordare în prvința identificării accentului o face Dindelegan [16]. Analizând gramatica limbii române ajunge la concluzia că accentul lexical nu este previzibil și astfel descurajează orice încercare de a determina reguli, avantajând învățarea supervizată.

Studile arată că problema despărțirii în silabe și a poziționării accentului prin intermediul unei abordări bazate pe dicționar s-au făcut cel mai mult pe foneme. Fonemele sunt o reprezentare a ortografiei cuvîntului care pot fi pasate la o componentă de sintetizare a vorbirii. Dar metodele sunt valide de asemenea, și pentru utilizarea literei în loc de fonem.

3.5. Construcția setului de date

Pentru a putea crea un sistem complet și de asemenea, pentru a putea realiza învățarea și testarea e nevoie de date. Pentru limba română date relevante și de folos nu sunt atât de ușor de găsit precum cele din limba engleză, spre exemplu. Să nu uităm că accentul nu se poziționează explicit la scriere, iar pentru o învățare corectă sistemul trebuie să aibă cunoștință și despre acest aspect. Apoi, o bază de date care să conțină cuvinte corect despărțite în silabe ar fi extrem de utilă. La fel ca și pentru accent, sistemul trebuie să învețe corect despărțirea în silabe pentru a face o predicție cât mai corectă. Sistemul bazat pe reguli nu presupune prea multe constrângeri, acesta aplicând șabloane direct pe cuvinte. Ce e important aici, însă, e o analiză a părții de vorbire – o trăsătură suplimentară ce poate fi necesară în problema accentului.

Pe de altă parte, limba română presupune scrierea cu diacritice, ceea ce înseamnă că sistemul trebuie să ia în considerare și acest aspect. Seturile de date trebuie să conțină deci și cuvinte cu diacritice.

Studiul bibliografic asupra seturilor de date[17], atrage atenția asupra a două dicționare descrise de către Ana Maria Barbu în articolul său [12]. Acestea conțin date și informații privind: forma canonică a cuvântului, vocalele accentuate sau silabele cuvântului. *RoSyllabiDict* conține aproximativ 550.000 de intrări-cuvinte cu informații privind despărțirea în silabe și poziția accentului, iar *RoMorphoDict* aproximativ 780.000 de cuvinte cu informații morfologice despre fiecare cuvânt.

Astfel, pot fi create seturile de antrenare utilizate în sitemele bazate pe învățare cât și seturile de date pe care să testăm sistemele.

Construcția presupune o analiză atentă a datelor și a trăsăturilor ce pot fi extrase, pentru a putea utiliza eficient și eficace algoritmii de *Machine Learning* . Seturile de date utilizate ca și modele în antrenare trebuie să fie corect construite, dar mai ales datele să fie relevante pentru ceea ce se dorește a se învăța.

4. Analiză şi Fundamentare Teoretică

Acest capitol face o analiză generală a aspectelor teoretice care facilitează înțelegerea proiectării și implementării soluției alese. Aspectele generale prezentate în capitolul anterior sunt referite împreună cu metodele utilizate pentru a îndeplini obiectivele acestei lucrări. Secțiunea 4.1 arată modelul general al întregii aplicații descriind modulele, în timp ce secțiunea 4.2 descrie structura aplicației pentru procesul de poziționare a accentului.

Soluția dezvoltată implică ambele sarcini: atât identificarea accentului cât și despărțirea în silabe, dar conform experimentelor, aceste probleme se pot trata și independent. În continuare am să prezint modelul general al sistemului în care se integrează cele două module dar evidențiez soluția aleasă pentru predicția accentului, referindu-mă la problema silabificării în măsura în care aceasta este necesară.

4.1. Model conceptual al aplicației

Obiectivul principal al sistemului de procesare al limbajului natural este acela de a identifica silabificarea și accentul cuvintelor din limba română, bazându-se pe experiențe anterioare similare.

O abordare condusă de date oferă mai multe avantaje, cu condiția să existe o cantitate suficientă de formare și validare a datelor, ceea ce reprezintă cazul nostru
Aceste date - cuvinte pentru care accentul și silabificarea se cunosc deja- sunt obținute prin procesarea dicționarelor [12] descrise în capitolul anterior: *RoSyllabiDict*, respectiv *RoMorphoDict*. Așadar, ceea ce se obține de aici se utilizează în antrenare pentru cele două module din următoarea etapă. Pe baza acestui model, noile date de la intrare sunt clasificate, pentru ca apoi datele obținute să fie tratate ca rezultat final.

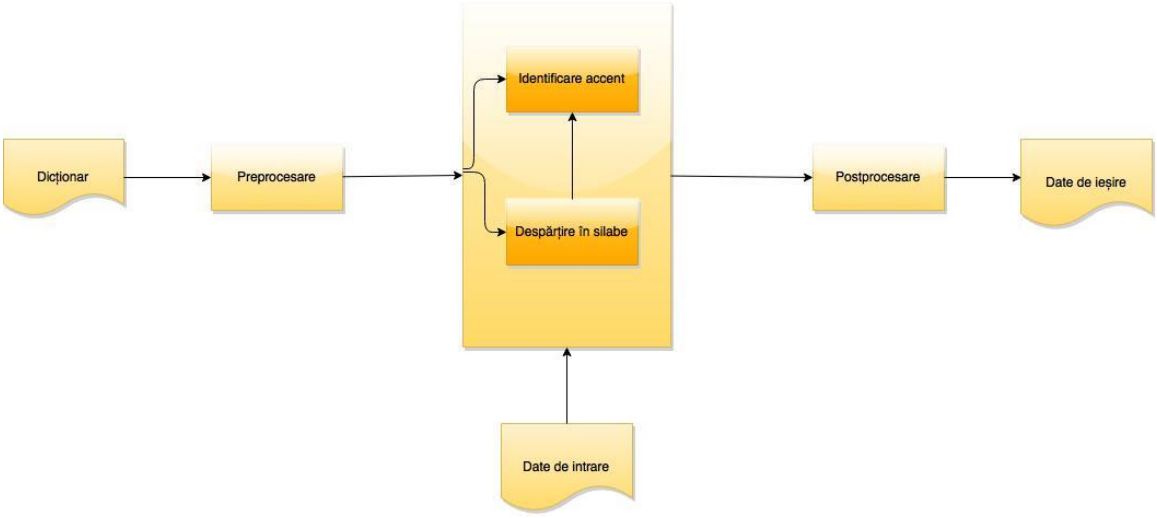


Figura 4.1 Modelul soluției aplicației

Cele două module de identificare a accentului și despărțire în silabe sunt integrate în același sistem unic și au în cea mai mare parte aceași parcurgere în sistem. Fiind două probleme cu trăsături diferite, comportamentul acestora diferă chiar și pentru același model de *Machine Learning*.

Pe de altă parte, pentru ca problema să devină și mai interesantă am luat în considerare și cazul în care rezultatele procesului de silabificare devin trăsături (intrări) în sub-sistemul de identificare al accentului, întreg sistemul devenind astfel un pipeline în care accentul depinde de despărțirea în silabe. Detalii specifice despre modelul pentru identificarea accentului vor fi descrise în paragrafele următoare.

În figura 4.1 este reprezentată o schemă generală a soluției pentru ambele sarcini.

4.2. Modelul sistemului pentru identificarea accentului

Modelul pentru procesul de identificare al accentului diferă față de cel pentru silabificare prin trăsăturile extrase, instanțele de antrenare, respectiv cele de testare. Pe de altă parte, această problemă este una extrem de sensibilă datorită lipsei accentului lexical în scrierea limbii literare. Excepțile provocate de către lipsa/prezența/poziționarea greșită a accentului într-un cuvânt au un efect mare în predicția instanțelor. Modelul soluției tratează în mod specific problema extragerii caracteristicilor unui cuvânt și o modelează în așa fel încât vectorul de trăsături construit să poată clasifica instanțele cât mai corect.

Figura 4.2 prezintă soluția propusă pentru această sarcină, urmând ca detaliile și analiza fiecărui modul să fie discutate în continuare.

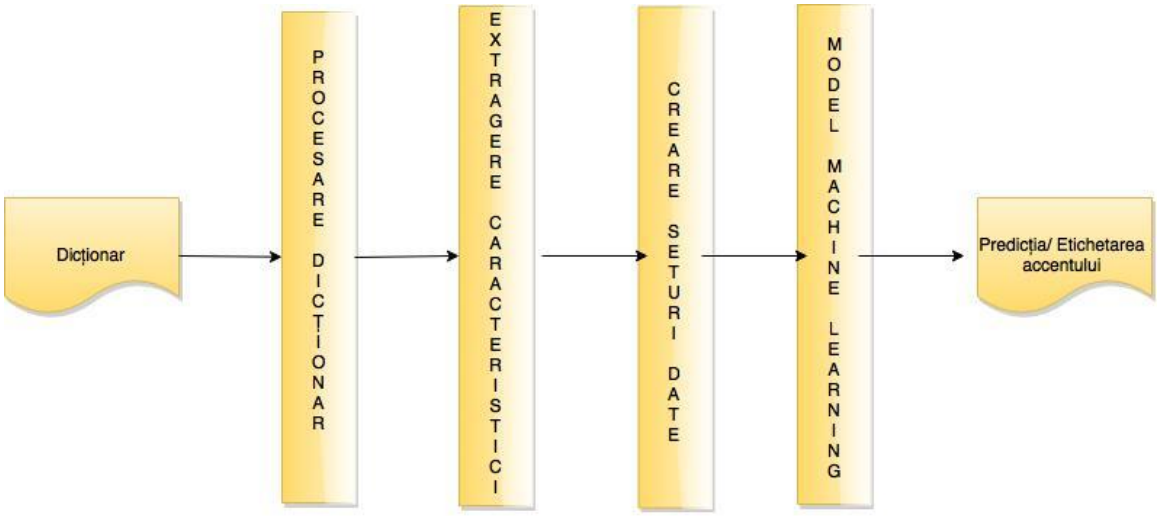


Figura 4.2. *Modelul pentru procesul de identificare al accentului*

4.3. Componente ale modelului pentru identificarea accentului

4.3.1. Procesarea dicționarului.

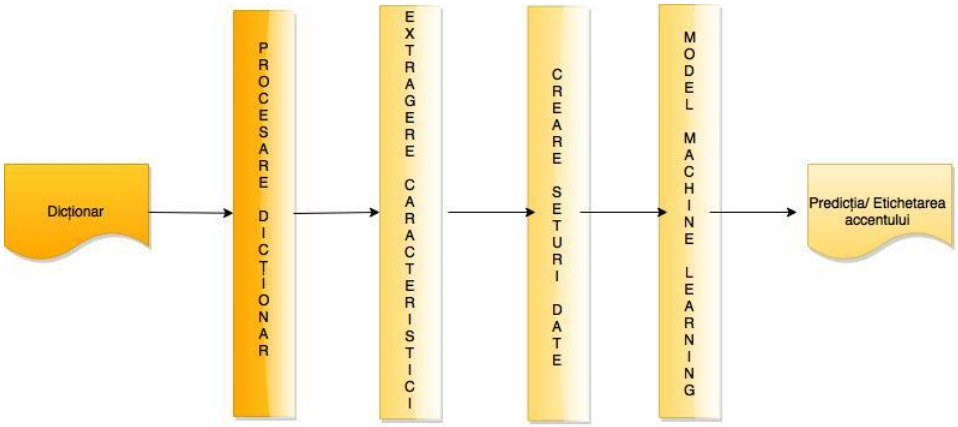


Figura 4.3 *Modulul de procesare al dicționarului*

Dintre cele două dicționare menționate, utilizăm doar unul și anume RoSyllabiDict. Acesta conține informațile cele mai relevante dar și suficiente pe care le putem utiliza ca bază în implementarea sistemului: forma inflexată a cuvântului, poziția accentului, despărțirea în silabe. Acestea devin trăsături care se învață pentru viitoarele predicții, detaliate ulterior. RoMorphoDict ne este de mai puțin ajutor venind cu informații referitoare la morfologia cuvintelor.

RoSyllabiDict conține 525,534 de cuvinte – forme inflexate- pe care le-am analizat. Dintre acestea, am găsit aproximativ 15,000 de excepții. Un procent de 68,4% din totalul excepților arată două posibilități de despărțire în silabe - corecte amândouă, 14,2% cuvinte scrise la fel dar cu partea de vorbire diferită, 9,2% omonime din care un număr de 1392 de cuvinte au aceași parte de vorbire dar accent diferit și 109 aceași parte de vorbire dar silabificare diferită. Figura 4.2 prezintă o clasificare a excepților din dicționar.

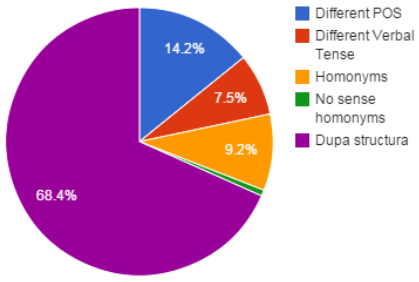


Figura 4.4 *Clasificarea excepților din RoSyllabiDict*

Majoritatea oamenilor nu utilizează diacritice în scrierea limbii literare. Dicționarul conține și cuvinte în forma corectă cu diacritice – din totalul cuvintelor, 254,968 sunt cu diacritice. Pentru a rezolva această problemă am analizat următoarele posibile situații:

- Cuvinte care nu au diacritice:
  - Număr de cuvinte : 270,566
  - Nu presupune nici o problemă
- Cuvinte care au forma fără diacritice dar nu corespund la un cuvânt existent
  - Număr: 234,464
  - Problemă: *păs**a**re* și *p**a**săre* au accentul poziționat diferit, iar dacă dispar diacriticele amândouă se transformă în același cuvânt care nu există în dicționar
- Cuvinte care au forma fără diacritice dar corespund la un cuvânt existent și silabificarea e la fel
  - Număr:18,873
  - Nu presupune nici o problemă
- Cuvinte care au forma fără diacritice dar corespund la un cuvânt existent și silabificarea nu e la fel
  - Număr de cuvinte:1631
  - Majoritatea sunt catalogate ca și excepții de la regulile de despărțire în silabe.

Trebuie menționat faptul că dicționarul a fost procesat și curățat de eventualele zgomote (caracterele străine - # º-). De asemenea, cuvintele monosilabice au fost eliminate, ele neavând nici o relevanță pentru vreo-una dintre sarcini. Nu aduc nici o informație utilă în plus care să ajute sistemul în performanță.

Totodată, dicționarul a fost utilizat în totalitate, creând atât seturi de date pentru antrenare cât și pentru testare.

4.3.2. Extragerea trăsăturilor și construcția vectorului de trăsături

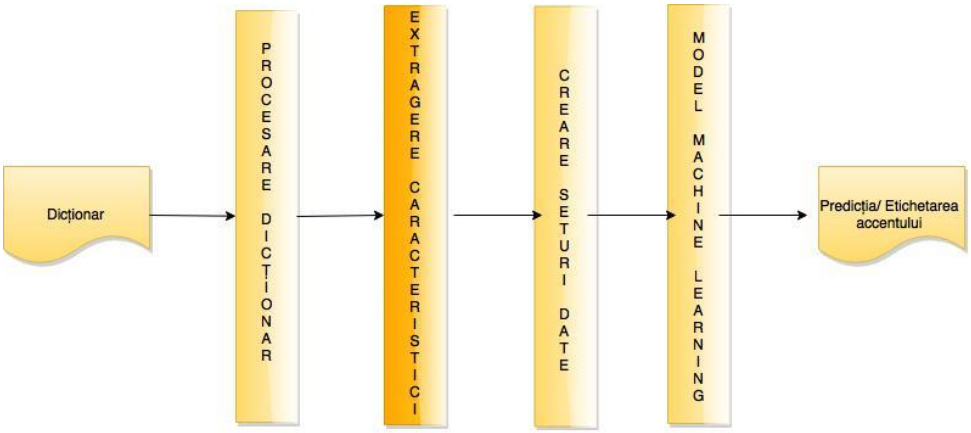


Figura 4.5 *Extragerea caracteristicilor (+crearea vectorului de trăsături)*

Având experiență deja cu problema alegerii trăsăturilor și clasei pentru silabificare, pentru identificarea accentului am pornit de la ideea lui Susan Barlett [14] și Qing Dou [15], utilizând în prima fază SVM și RF pentru predicție. Așa cum am descris în capitolul anterior, trăsăturile formează un vector pentru fiecare instanță, iar pe baza acestor vectori algoritmul separă instanțele pozitive de cele negative.

Extracția elementelor vectorului submodulului fost dezvoltat folosind capacitatea selectare a trăsăturilor furnizate de Weka și este un pas fundamental al fluxului de clasificare. Practic, toate tehnicile de prelucrare efectuate de celelalte nivele sunt aplicate utilizând caracteristicile extrase de acest submodul, acesta fiind principalul motiv pentru care integrarea cu Weka a fost una dintre primele sarcini efectuate.

Vectorul de trăsături a fost modelat într-un proces iterativ, pe baza experimentelor și performanțelor. În mod specific, pentru fiecare am luat ca și intrare un (sub)set etichetat iar ca și ieșire instanțele și trăsăturile corespunzătoare. Pentru a permite efectuarea unui proces de clasificare regular, am transformat fiecare cuvânt într-un set de instanțe – cu mărimea egală cu lungimea cuvântului – și am stabilit eticheta ce reprezintă clasa vectorului - accentuat/neaccentuat (adică dacă instanța este accentuată sau nu).

Setul de instanțe este construit pe baza n-gram-elor[18]. Acestea sunt secvențe continue de n elemente dintr-o anumită secvență de text. Elementele pot fi foneme, silabe, litere sau cuvinte în funcție de aplicație. N-grame sunt de obicei colectate de la un corpus de text sau discurs (din vorbire). N-gramele de lungime 1 se numesc unigrame, cele de lungime 2 bigrame (se mai utilizează și digrame), de lungime 3 trigrame ș.a.m.d.

Figura următoare arată descompunerea în bigrame și trigrame, cazul unigramelor fiind clar – cât un element. Primul exemplu este dat pe o propoziție unde gram-ul este reprezentat de cuvânt. Practic bigramele se compun din 2 cuvinte, cel curent și cel de imediat lângă (vecinul). Construcția se face de la dreapta la stânga dar la fel de bine se poate face și invers, de la stânga la dreapta. În cazul trigramelor se iau 3 cuvinte: cuvântul curent, primul cuvânt vecin din stânga, al doilea cuvânt vecin din stânga. Dacă exemplul s-ar continua și pentru quad-grame (4 grame), structura ar arăta așa: cuvântul curent, primul cuvânt vecin din stânga, al doilea cuvânt vecin din stânga, al treilea cuvânt din stânga.

Cazul B exemplifică pe modul general, astfel încât acest mecanism se poate aplica pentru orice tip de gram: literă, cuvânt, silabă, etc.

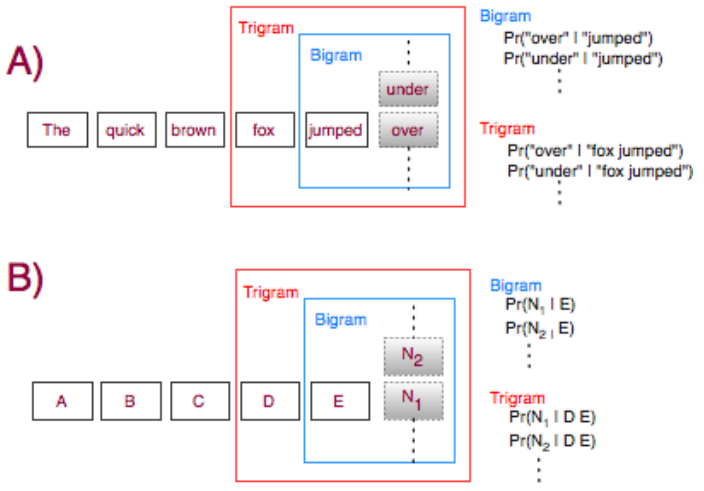


Figura 4.6 *Construcția n-gramelor.*

În cazul nostru, un gram este reprezentat de către un caracter dintr-un cuvânt. De exemplu, cuvântul "TEXT" este descompus după regula n-gramelor, astfel:

- Bi-gram: _T, TE, EX, XT, T_
- Tri-gram: _TE, TEX, EXT, XT_, T_ _
- Quad-gram: _TEX, TEXT, EXT_, XT_ _, T_ _ _

Am adoptat o formalizare bazată pe mecanismul n-gramelor pentru construcția vectorului, astfel: fiecare literă-vocală a cuvîntului este o instanță descrisă de un set de trăsături constând într-o secvență (fereastră) de vecini de lungime zece, cinci la stânga și cinci la dreapta pentru litera curentă. Adăugarea unui caracter special (*) a fost necesară pentru a completa poziția din vector în cazul literelor cu secvențe mai mici (au mai puțin de 5 vecini pe fiecare parte). Luăm ca și instanțe doar vocalele pentru că în limba română accentul nu poate fi pus pe consoane, astfel seturile de date se reduc considerabil.

Figurile următoare exemplifică trăsăturile pentru vector utilizând cuvântul "înțelegi". Clasa este marcată de 1 - accentuat sau 0 – neaccentuat.

Litera	*Trăsături*	*Clasa*
î	* * * * * î n ţ e l e	0
e	* * î n ţ e l e g i *	0
e	î n ţ e l e g i * * *	1
i	ţ e l e g i * * * * *	0

Figura 4.7 *Trăsături pentru cuvântul "înțelegi"*

î	*	*	*	*	*	*î*	*n*	*ţ*	*e*	*l*	*e*	*0*

Figura 4.8 *Vectorul de trăsături pentru cuvântul "înțelegi"utilizând unigrame*

Figura 4.8 ilustrează vectorul de trăsături pentru unigrame. Pe prima poziție se află litera luată ca și instanță, urmează cinci poziții cu unigrame pentru litera î - în cazul de față fiind prima literă nu are vecini în stânga și completăm cu *- apoi unigramul curent, urmată de alte 5 poziții cu vecinii din dreapta. Vectorul se încheie cu clasa 0 care ne spune că instanța nu este accentuată.

În prima fază, după cum se poate observa și din figura de mai sus, vecinii literei curente au fost luați ca unigrame. Am luat în considerare și cazul bigramelor, descris în figura de mai jos.

Litera	*Trăsături*	*Clasa*
î	** ** ** ** ** î în nţ ţe el le	0
e	** ** în nţ ţe e el le eg gi **	0
e	în nţ ţe el le e eg gi ** ** **	1
i	ţe el le eg gi i ** ** ** ** **	0

Figura 4.9 *Trăsături pentru cuvântul "înţelegi" folosind bigrame*

î	**	**	**	**	**	*î*	*în*	*nţ*	*ţe*	*el*	*le*	*0*

Figura 4.10 *Vectorul de trăsături pentru cuvântul "înţelegi"utilizând bigrame*

Figura 4.10 ilustrează vectorul de trăsături pentru bigrame. Pe prima poziție se află litera luată ca și instanță, urmează cinci poziții cu unigrame pentru litera î - în cazul de față fiind prima literă nu are vecini în stânga și completăm cu *- apoi instanța pentru care se crează vectorul, urmată de alte 5 poziții cu vecinii din dreapta. Vectorul se încheie cu clasa 0 care ne spune că instanța nu este accentuată.

Se construiește câte un vector pentru fiecare instanță a cuvântului. Așa cum am menționat mai sus, în cazul accentului, se iau în calcul doar vocalele, deci un cuvânt va avea un număr de vectori egal cu numărul vocalelor din cuvânt.

Susan Barlett[14] spune că o îmbunătățire a rezultatelor nu se mai vede de la tri-grame în sus. Noi nu am mers mai departe datorită rezultatelor de predicție dar și de performanță mai slabe obținute pentru bi-grame și descrise în capitolul 6.

În schimb, am adaptat problema astfel încăt trăsăturile vectorului pentru accent să utilizeze informații de la modulul de silabificare. Am adăugat două trăsături în plus față de modelul vectorului cu unigrame pentru a vedea cum se comportă predicția. Aceste trăsături sunt: numărul de silabe al cuvîntului din care provine litera pentru care se construiește vectorul, respectiv poziția literei în cuvânt de la stânga la dreapta (a câte-a literă e în cuvânt?). Modelul vectorului e cel bazat pe unigrame. Figura 4.11 ne arată trăsăturile pentru construcția vectorului și figura 4.12 descrie vectorul de trăsături .

Litera	*Trăsături*	*Clasa*
î	3 1 * * * * * î n ţ e l e	0
e	3 4 * * î n ţ e l e g i *	0
e	3 6 î n ţ e l e g i * * *	1
i	3 8 ţ e l e g i * * * * *	0

Figura 4.11 *Trăsături pentru cuvântul "înţelegi" folosind 2 trăsături în plus obținute prin procesul de silabificare*

După cum se poate observa din figura 4.11, pe prima poziție a vectorului se află informația cu privire la numărul de silabe, iar pe a doua, cifra ce reprezintă poziția instanței în cuvînt. Trebuie precizat că instanțele sunt reprezentate doar de vocale, dar trăsăturile țin cont de toate literele cuvântului.

î	*3*	*1*	**	**	**	**	**	*î*	*în*	*nț*	*țe*	*el*	*le*	*0*

Figura 4.12 *Vectorul de trăsături pentru cuvântul "înțelegi"utilizând 2 trăsături în plus obținute prin procesul de silabificare.*

4.3.2.1. Obținerea trăsăturilor prin procesul de silabificare

Modulul de silabificare l-am gândit în două variante diferite. Prima variantă a fost construirea unui sistem bazat pe reguli care implementează regulile de despărțire în silabe. Pentru cuvintele care se mapează la aceste reguli, procesul decurge normal. Problema vine în discuție pentru acele cuvinte ce sunt excepții de la regulă menționate în capitolul 3. Clasificarea nu se face corect în mod uniform (tabel 4.1), nu putem extrage informații corecte pe care să le utilizăm în construcția vectorului de trăsături pentru accent, ceea ce ne-a determinat să schimbăm strategia. Testele au fost efectuate pe un număr de 193 cuvinte cu 407 silabe în total.

Base case	**Cuvinte testate**	**Identificate și aplicate corect**	**Aplicate incorect (nu ar fi trebuit să se aplice)**	**Nu le-a găsit (missed)**
I.1 V – **C**V	193	406	9	1
I.2 V**C** – **C**V	193	66	3	11
I.3 **C** – **CC**	193	9	0	4
I.4 **C** – **CCC**	193	1	0	0
II.1 **V**-**V**(S)	193	11	3	34
II.2 (S)**V**-**S**V	193	8	2	2

Tabel 4.1 *Rezultate obținute pentru sistemul bazat pe reguli*

După cum se poate observa și din tabel, pentru regulile de despărțire în silabe bazate pe vocale, din cele 45 de silabe care merg pe cazul V-V(S), doar 11 au fost identificate și aplicate corect, 34 dintre ele negăsindu-le deloc.

A doua soluție e cea bazată pe învățare, utilizând aceași structură a vectorului de trăsături ca și pentru accent. Pentru a verifica ce algoritm de învățare face predicții mai bune am utilizat mai multe variante de algorimti. Rezultatele l-au pus pe primul loc, pe RandomForest. Tot pe baza experimentelor efectuate pentru silabificare am luat decizia de a aplica în prima fază algoritmul RF și pentru problema accentului.

Figura 4.6 arată vectorul de trăsături pentru cuvîntul "înțelegi" utilizat în predicția silabelor. Structura vectorului e asemănătoare cu cea a accentului cu diferența că pe prima poziție punem litera curentă, iar pe a doua tipul literei- vocală sau consoană. În rest, secvența de vecini e creată pe modelul unigramelor. În figura de mai jos, clasa e

notată cu no – *Not Boundary* și yes – *Boundary*; adică dacă litera e urmată de simbolul silabificării "-" sau e ultima din cuvânt, aceasta este la limită și determină o silabă.

Litera	*Trăsături*	*Clasa*
î	î l * * * * * î n ţ e l e	no
n	n 0 * * * * î n ţ e l e g	yes
ţ	ţ 0 * * * î n ţ e l e g i	no
e	e l * * î n ţ e l e g i *	yes
l	l 0 * î n ţ e l e g i * *	no
e	e l î n ţ e l e g i * * *	no
g	g 0 n ţ e l e g i * * * *	no
i	i l ţ e l e g i * * * * *	yes

Figura 4.13 *Trăsături pentru cuvântul "înțelegi" pentru problema silabificării*

î	*3*	*1*	**	**	**	**	**	*î*	*în*	*nț*	*țe*	*el*	*le*	*0*

Figura 4.14 *Vectorul de trăsături pentru cuvântul "înțelegi"pentru procesul de silabificare*

Tabelul 4.2 prezintă rezultatele obținute în urma clasificării, precum și informații precum precizia și rata *false-positive.* Fiecare model antrenat conține 4295 de cuvinte, aproximativ 42,500 de instanțe. Seturile de date sunt diferite din punct de vedere al cuvintelor utilizate.

După cum se poate observa din tabel, Random Forest face cele mai bune predicții și are și o performanță mai bună din punct de vedere al timpului de clasificare comparativ cu SMO care pe lângă diferența de 3%, are un timp de excuție mult mai mare.

Model	**Classifier**	**Correct Classification**	**Precision**	**FP Rate**
Set01	Random Forest	99.46%	99.5%	0.50%
	SMO	96.27 %	96.3 %	4.30%
	Naive Bayes	87.03 %	87.5 %	12.20%
	Ada Boost	80.92 %	81.3 %	18.80%
Set02	Random Forest	99.00 %	99.0%	1.00%
	SMO	96.04%	96.1 %	4.50%
	Naive Bayes	87.09 %	87.6%	12.10%
	Ada Boost	80.92 %	81.3%	18.80%
Set03	Random Forest	98.93 %	98.9%	1.10%
	SMO	96.02 %	96.0%	4.40%
	Naive Bayes	87.13 %	87.7%	12.00%
	Ada Boost	80.92 %	81.3%	18.80%

Tabel 4.2 *Rezultate obținute în urma aplicării diferiților algoritmi pentru predicția silabelor.*

4.3.3. *Crearea seturilor de date pentru antrenare (și testare)*

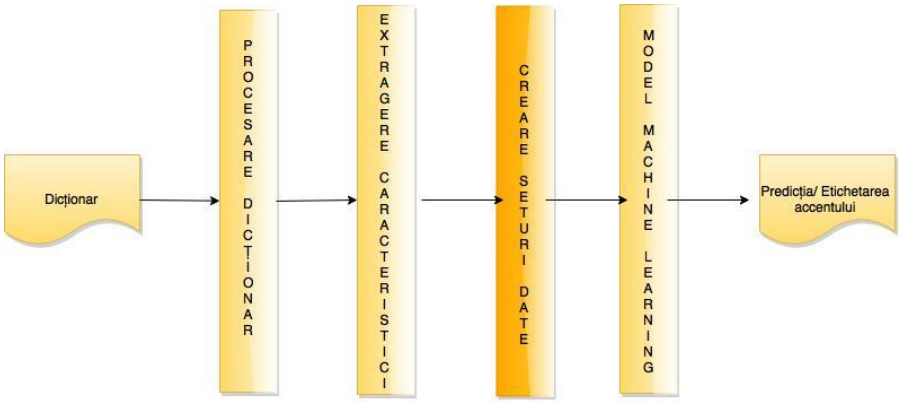


Figura 4.15 *Crearea seturilor de date*

Seturile de date create trebuie să respecte un format și anume, formatul cu extensia .arff. Capitolul 5 descrie necesitatea acestui format și implementarea tehnologiilor utilizate. Un fișier ARFF (*Atribut-Relation File Format*) este un fișier text ASCII. Diferența o reprezintă conținutul care descrie o listă de instanțe partajate pentru un set de atribute. Fișierele ARFF au fost elaborate de *Machine Learning Project* din cadrul Departamentului de Informatică de la Universitatea din Waikato[19] pentru a fi utilizate cu software-ul de ML Weka, despre care discut în detaliu în capitolul 5.

```
@relation 'Set01Accent'

@attribute Letter {a,ă,â,e,î,i,o,u}
@attribute UniPrev5 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniPrev4 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniPrev3 {*,a,ă,â,,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniPrev2 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniPrev1 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute Unigrahm {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniAfter1 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniAfter2 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniAfter3 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniAfter4 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute UniAfter5 {*,a,ă,â,b,c,d,e,f,g,h,î,i,j,k,l,m,n,o,p,q,r,s,ș,t,ț,u,v,w,x,y,z}
@attribute Class {0,1}

@data

a,*,*,*,*,*,a,b,a,c,ă,*,0
a,*,*,*,a,b,a,c,ă,*,*,*,1
a,*,*,*,*,*,a,b,a,n,d,o,0
a,*,*,*,a,b,a,n,d,o,n,i,0
o,a,b,a,n,d,o,n,i,c,e,*,1
i,a,n,d,o,n,i,c,e,*,*,*,0
e,d,o,n,i,c,e,*,*,*,*,*,0
a,*,*,*,*,*,a,b,a,ț,i,i,0
```

Figura 4.16 *Fișier .arff*

[19] http://www.cs.waikato.ac.nz/

Menționez acum acest aspect, deoarece seturile de date trebuie procesate pentru a ajunge din formatul de la intrarea sistemului .txt în format .arff.

Structura fișierului cu extensia .arff este relativ simplă, din două secțiuni: una este repreznțată de *header* și cealaltă de *data*. Secțiunea de header trebuie să cupindă, următoarele:

- ***@ relation*** - Prima linie din fișier setează denumirea relației (setului de date)
- ***@attribute*** – lista de trăsături : denumirea și posibilele valori ale acestora; atribute pot fi oricâte. Ordinea este importantă pentru că trebuie respectată de către instanțele din secțiunea de *data*
- ***@attribute Class*** **–** nu poate lipsi niciodată, fiind atributul care dă clasa de predicție

Secțiunea de ***@data*** cuprinde pe câte o linie o instanță, iar fiecare instanță are un vector de trăsături construit pe baza atributelor setate în header. De exemplu secvența ***a,*,*,*,*,*,a,b,a,c,ă,*,0*** - reprezintă un vector de trăsături pentru instanța ***a*** **.** Dacă ne referim la al treilea atribut din secțiunea de declarații, atunci pentru această instanță, atributul îl găsim pe a treia poziție.

Aceste seturi create, după cum arată și figura de mai sus, provin din procesarea dicționarului. Se poate însă ca modelul antrenat – obținut din dicționar -să fie aplicat pe seturi de test altele decât cele din dicționar. Aceste seturi însă, pentru a putea fi procesate de către modelul de Machine Learning trebuie să treacă prin aceași pași începând cu extragerea caracteristicilor. Ce se pierde în acest caz e partea de procesare care s-a efectuat pe dicționar. S-ar putea ca alte fișiere să conțină zgomote și ne-trecând prin faza de preprocesare aceste aspecte să fie trecute cu vederea.

O altă observație, în cazul în care se utilizează alte fișiere e că acestea trebuie să fie de tipul .txt și să conțină doar text.

4.3.4. Modelul de ML

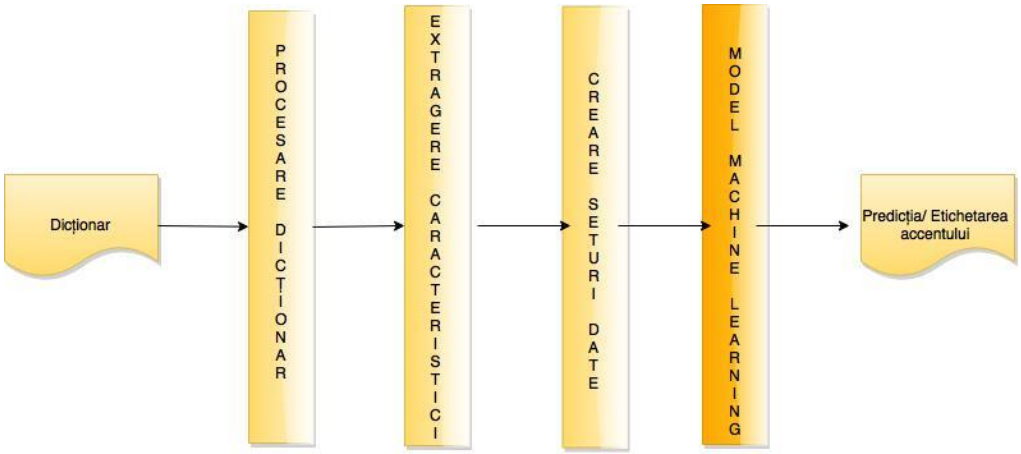


Figura 4.17 *Modulul de ML*

Modelul de Machine Learning aplică algoritmii de învățare supervizată pe seturile de date create.

Aici putem vorbi de mai multe cazuri:

1. Antrenăm un model după care îl aplicăm unui set de date de test pentru a prezice poziția accentului instanțelor din set, prin opțiunea de ***use supplied test*** .Această situație este permisă de aplicația integrată prin interfață acesteia; scopul aplicației integrate este de a obține predicția accentului și a silabificării.
2. Antrenăm un model alegând opțiunea de ***cross-validation****[19]cu setarea valorii pentru folds.* Prin acest proces, setul de date pe care îl dăm spre antrenare se împarte în două: o parte se utilizează ca și model de antrenare, iar cealaltă parte dată de valoarea fold-ului ca și set de test.
3. O altă opțiune e cea de ***percentage split***, setul de date pe care îl dăm spre antrenare se împarte în funcție de procentajul setat. O parte se antrenează, iar cealaltă se testează pe baza modelului antrenat.
4. Pentru a obține doar modelul antrenat, opțiunea disponibilă este cea de ***use training set.***

Cazurile 2,3,4 nu sunt disponibile prin intermediul aplicației integrate și de fapt, nu acesta este scopul aplicației. Aceste cazuri pot fi întâlnite , însă pe fiecare modul în parte. În capitolul 5 explic în detaliu cum se pot utiliza aceste opțiuni și prin intermediul cui. Rolul acestora e de a ajuta în obținerea statisticilor privitoare la predicție.

Cazul 1 e ceea ce ne interesează pentru obiectivul final și necesar pentru a putea prelucra rezultatele predicților prezentate în continuare.

4.3.5. Procesarea predicților

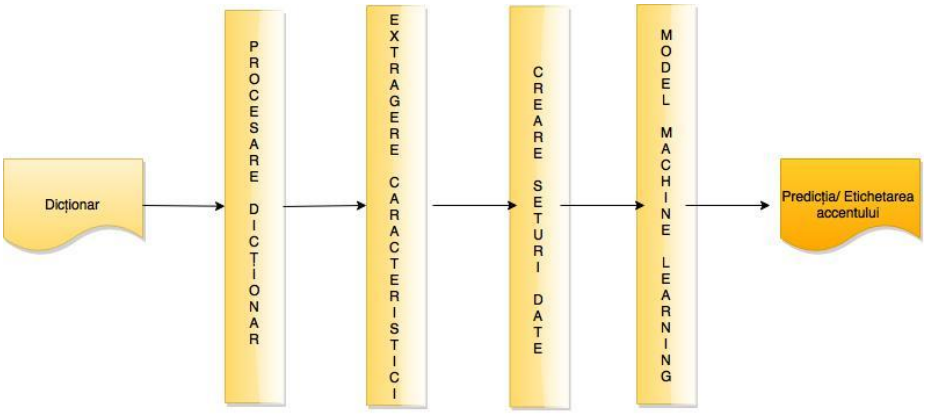


Figura 4.18 *Etichetarea accentului*

Ceea ce rezultă în urma aplicării algoritmilor de învățare sunt secvențe de 1 și 0, adică clasa prezisă pentru fiecare instanță. Din ceea ce se prezice se poate însă reface cuvîntul cu accentul poziționat pentru ca apoi utilizatorul să vadă rezultatele.

Pe de altă parte, pe lângă cuvântul clasificat se mai obțin și rezultate cu privire la acuratețe, precizie, clasificare corectă care interpretate ne ajută să ne dăm seama cât de bun este sistemul. Aceste rezultate pot fi punct de interes pentru utilizatorii-cercetători în domeniu.

5. Proiectare de Detaliu și Implementare

Capitolul 5 descrie în detaliu partea de proiectare și implementare a aplicației. Prima secțiune vorbește despre arhitectura sistemului, face o descriere generală a întregului sistem evidențind subsistemul de identificare al accentului. Secțiunea 5.2 detaliază partea de proiectare a aplicației prin diagrame, în timp ce secțiunea 5.3 arată implementarea sistemului pentru fiecare modul în parte. Ultima secțiune ilustrează tehnologiile utilizate în aplicație.

5.1. Arhitectura sistemului

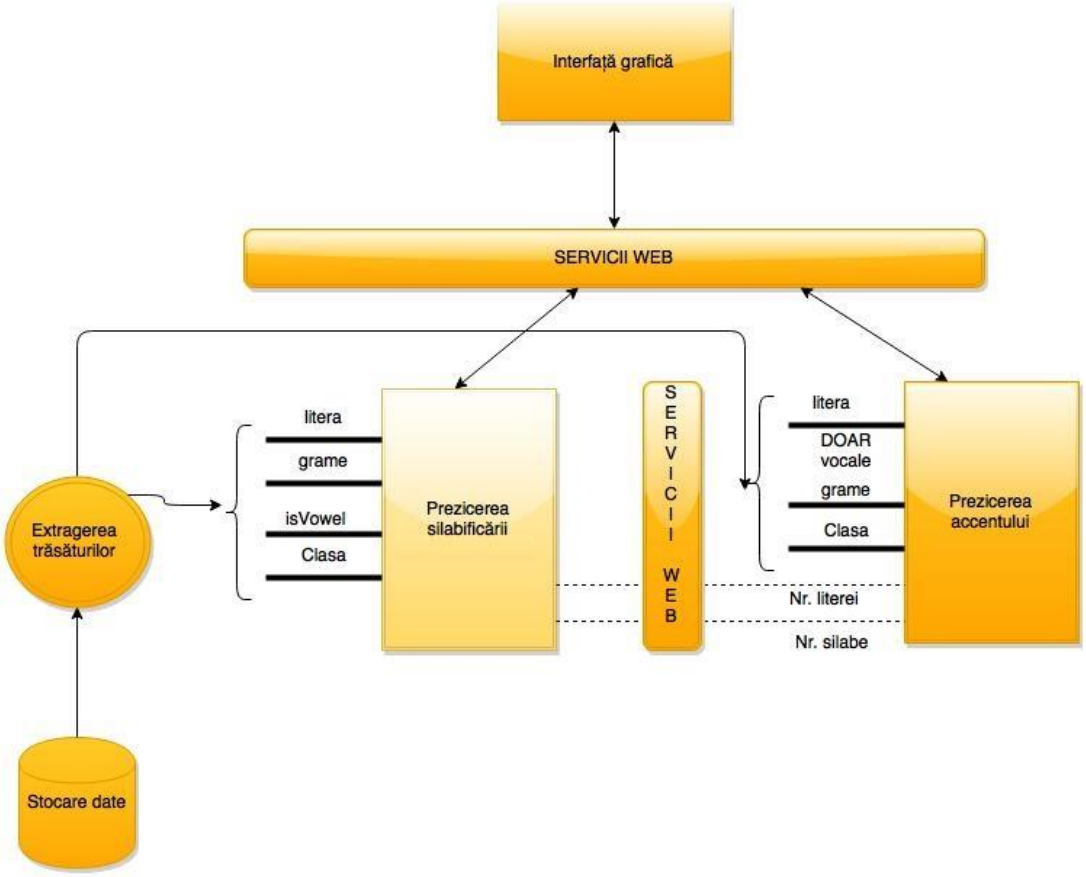


Figura 5.1 *Arhitectura sistemului. Intregrarea modulului de prezicere al accentului cu cel de silabificare*

Arhitectura sistemului este prezentată în figura de mai sus, arâtând integrarea modulului de prezicere al accentului cu cel de silabificare.

Stocarea datelor este dată de dicționar în sine, într-un fișier txt nu mai mare de 15MB. Datele sunt preluate și trecute prin procesul de extragere a trăsăturilor pentru a obține atributele necesare construirii fișierelor arff. Acestea sunt supuse antrenării, respectiv evaluării/testării pe baza vectorului de trăsături construit pentru fiecare modul. Rezultatele obținute sunt procesate și apoi trimise utilizatorului în interfața grafică prin servicii Web.

Cele două module, de prezicerea silabificării și prezicerea accentului cominică prin servicile Web, astfel: după obținerea predicției, modulul de silabificare îi transmite modulului de prezicere al accentului două trăsături utilizate de acesta în construirea celui de-al doilea tip de vector de trăsături: numărul de silabe al cuvântului din care face parte instanța și numărul literei din cuvânt pentru aceași instanță. Acest serviciu este utilizat doar atunci când folosim în construirea seturilor de date vectorul menționat anterior. Modulul de prezicere al accentului poate fi utilizat și independent de silabificare.

Implementarea am efectuat-o în două limbaje diferite: C#, respectiv Java. Pentru partea de predicție, librăria necesară este disponibilă doar în Java. Descriu acest aspect în detaliu în subcapitolele următoare. Figura 5.2 descrie arhitectura pentru identificarea accentului arătând și limbajul de programare în care este implementat fiecare modul.

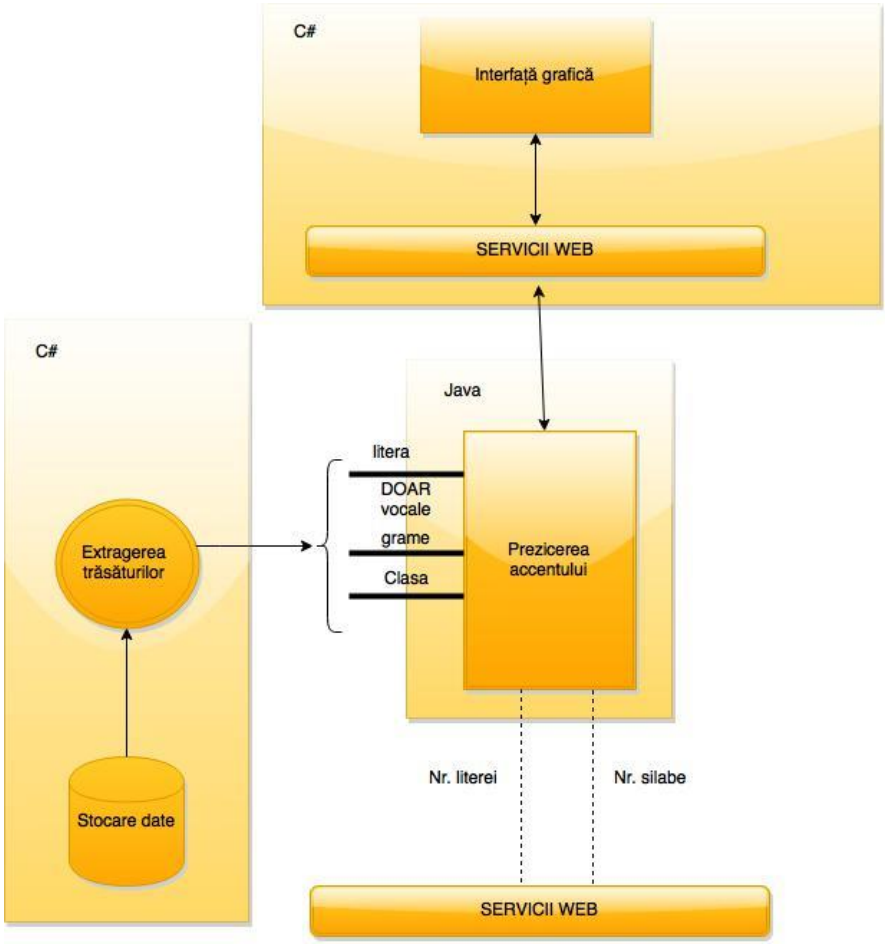


Figura 5.2 *Arhitecturta modulului de identificare al accentului și limbajul de programanre utilizat în implementarea submodulelor.*

În continuare voi descrie în detaliu partea de identificare a accentului și modulele aferente și necesare acestuia : stocare date, extragerea trăsăturilor, servicii Web și interfața grafică, după modelul prezentat în figura 5.2

5.2. Descrierea proiectării

Așa cum am menționat în secțiunea anterioară, am utilizat două limbaje de programare diferite pentru submodule. În Java, am implementat tot ce ține de clasificare: construirea fișierlor arff, antrenare, predicție, în timp ce pe partea de C#: parsarea datelor, extragerea trăsăturilor, procesarea rezultatelor și interfața grafică.

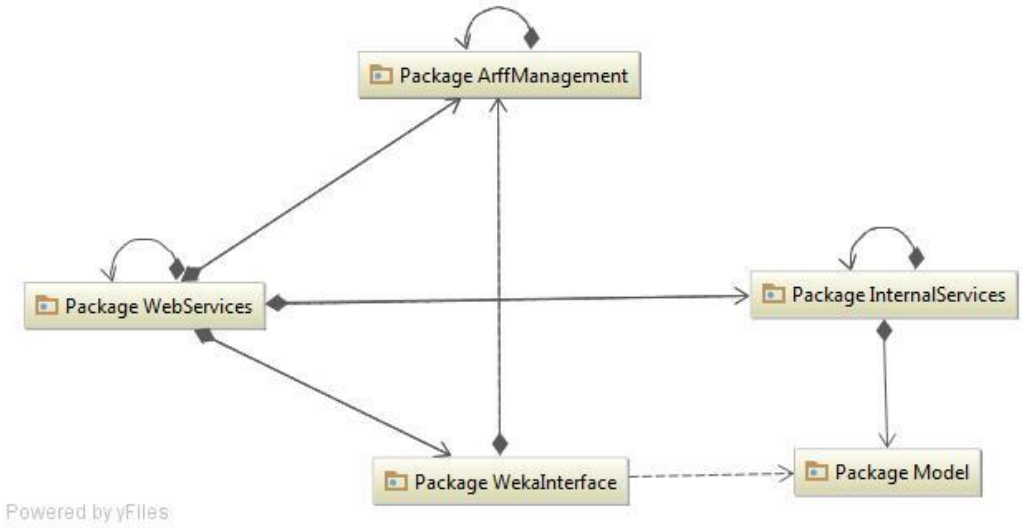


Figura 5.3 *Diagrama de pachete a modulului de predicție al accentului.*

Figura 5.3 descrie modulul implementat în Java printr-o diagramă de pachete. Pentru fiecare responsabilitate (funcționalitate) avem câte un pachet, astfel:

- ArffManagement – se ocupă de construirea fișierelor arff, procesarea pentru antrenare, respectiv testare și stocare pe baza trăsăturilor primite de la submodulul implementat în C# prin servicile Web.
- InternalService – are în responsabilitate partea de stocare: a modelelor antrenate, a fișierelor de test, de antrenare, selectarea modelelor antrenate în funcție de sesiunea în care s-a efectuat antrenarea.
- Model – are în prim plan algorimtmii de învățare
- WebServices - se ocupă de request-uri și response-uri între cele două subsisteme
- WekaInterface – utilizează librăria de Weka pentru clasificare

1. <u>ArffManagement:</u>
   Figura 5.4 prezintă diagrama de clase pentru pachetul Arff Management.
   - ArffParser.class – principala clasă a pachetului, face parsarea întregului fișier (crează fișierul final)
   - ArffStream.class – construiește fișierul arff pe bucăți.
   - ArffArgParser.class – finalizază construirea fișierului arff prin setarea claselor vectorului de trăsături îm funcție de scopul fișierului: antrenare sau testare.
   - ArffPathReceiver. Class – permite accesarea fișierelor salvate pe disk

- ArffLetters.class – are în responsabilitate operațile cu tipurile de litere existente.

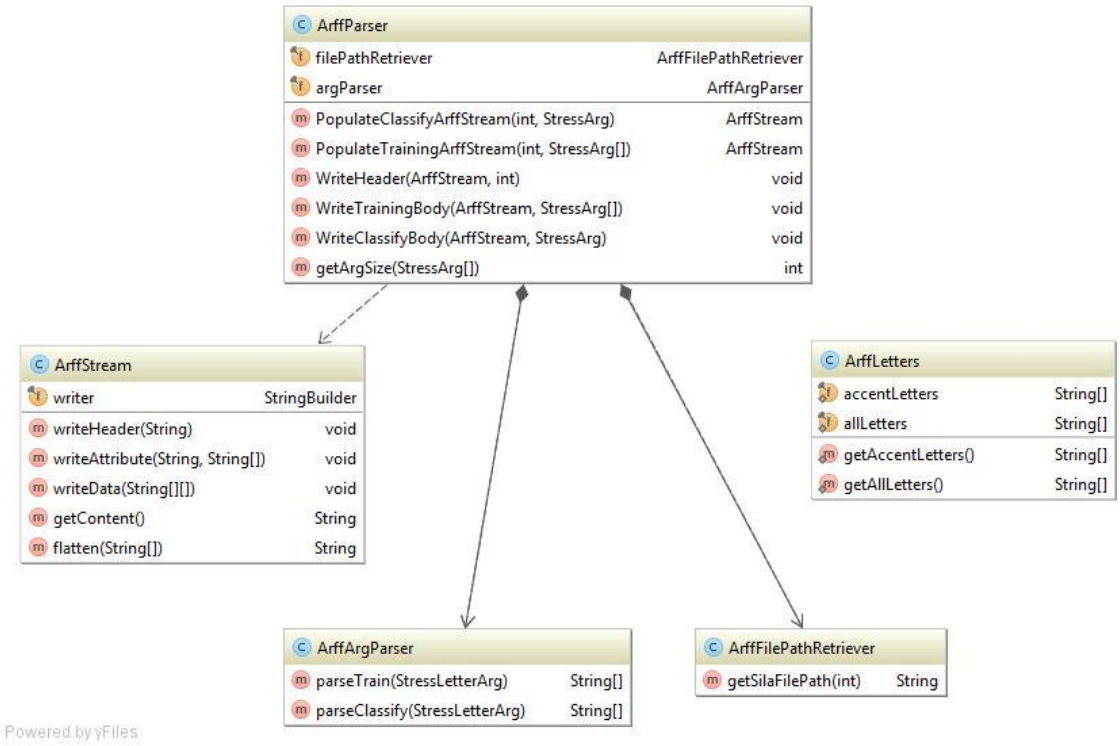


Figura 5.4 *Diagrama de clase pentru pachetul Arff Management*

2. Model:
   - Classifier.class – gestionează algoritmii de învățare
     - Pachetul este de fapt o extensie pentru utilizarea mai multor algoritmi în învățare.

Figura 5.5 *Diagrama de clase pentru pachetul Model*

3. InternalService:
   - FileManager.class - gestionează fișierele de intrare, respectiv ieșire
   - PathProvider.class – în funcție de sesiune, ia fișierul corespunzător de pe disk.

- ClassifierManager.class – gestionează fișierele antrenate de clasificatori, le salvează local pentru ca la următoarea antrenare cu același clasificator să încarce modelul deja existent.
- Cache.class – gestionează fișierele antrenate de clasificatori stocându-l sau dacă există, îl returnează

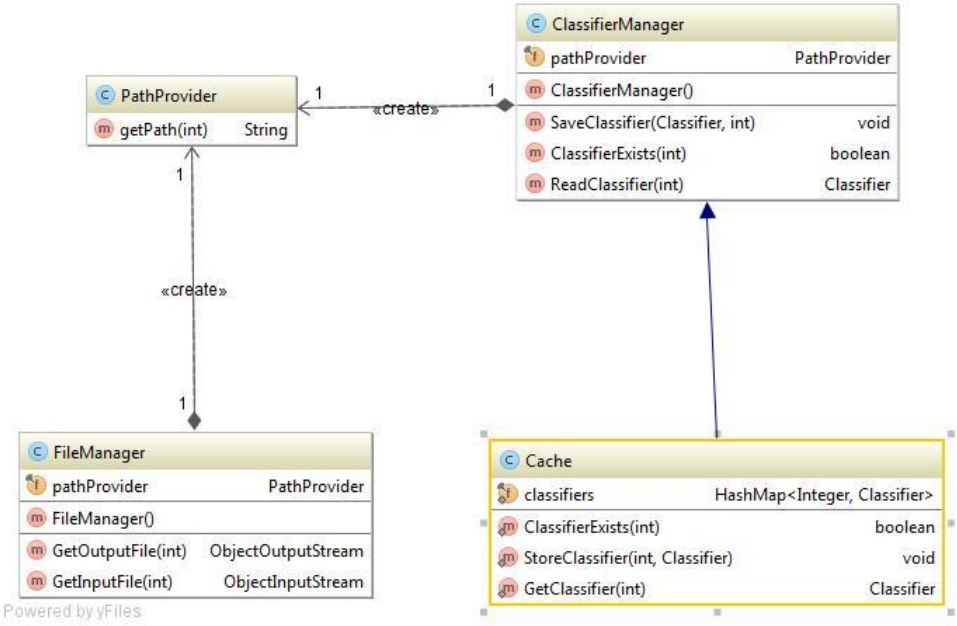


Figura 5.6 *Diagrama de clase pentru pachetul Internal Service*

4. WebService:
    - StressPredictionService.class – comunică prin servicii Web, transmițând rezultatele predicției (antrenare + testare)
    - StressLetterArg.class – gestionează trăsăturile primite tot prin servicii web pentru a le trimite mai departe pachetului de gestionare al fișierelor arff.
    - StressArf.class – construiește vectorul de trăsături

- Există 3 clase care tratează posibilele excepții apărute din cauza servicilor web, clasificării sau încărcării de fișiere. Astfel erorile nu se pierd, sunt gestionate și tratate.

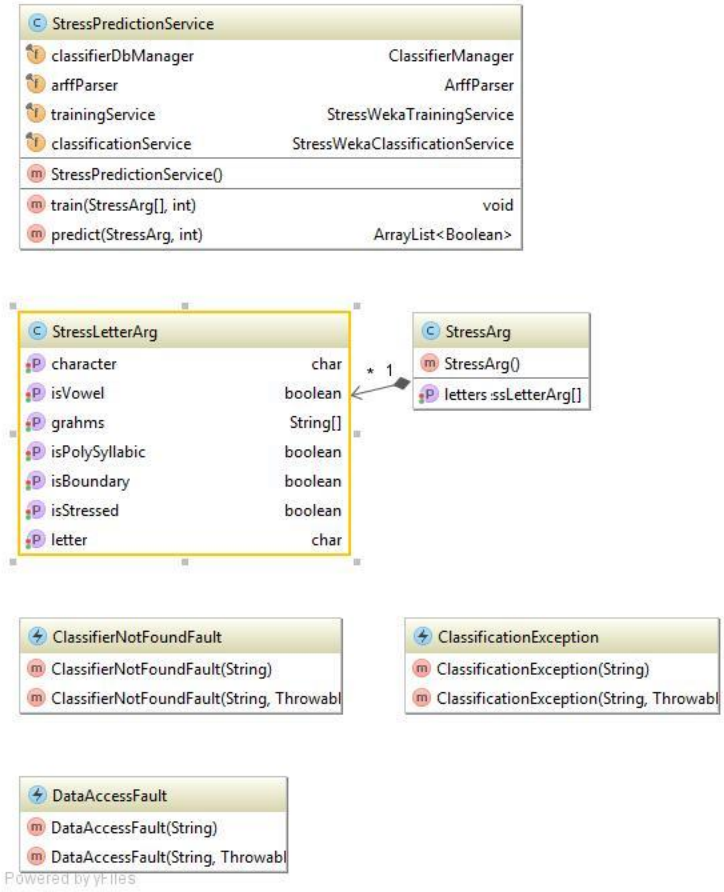


Figura 5.7 *Diagrama de clase pentru pachetul WebService*

5. WekaInterface:

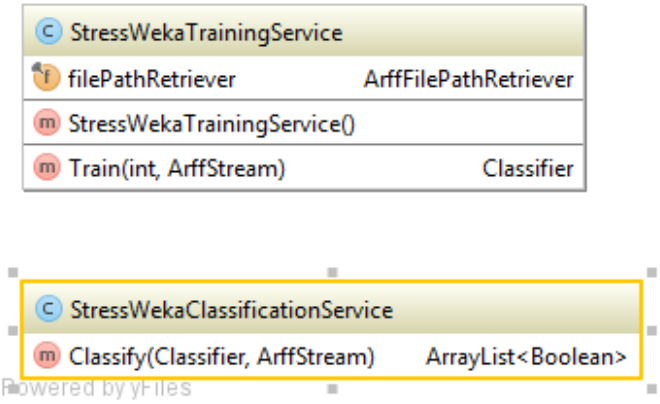


Figura 5.7 *Diagrama de clase pentru pachetul WekaInterface*

- StressWekaTrainingService.class –antrenează modelul
- StressWekaClassificationService.class – clasifică pe baza modelului antrenat, iar predicția obținută este trimisă prin servicii web spre submodulul implementat in C#, pentru procesare și afișare de rezultate.

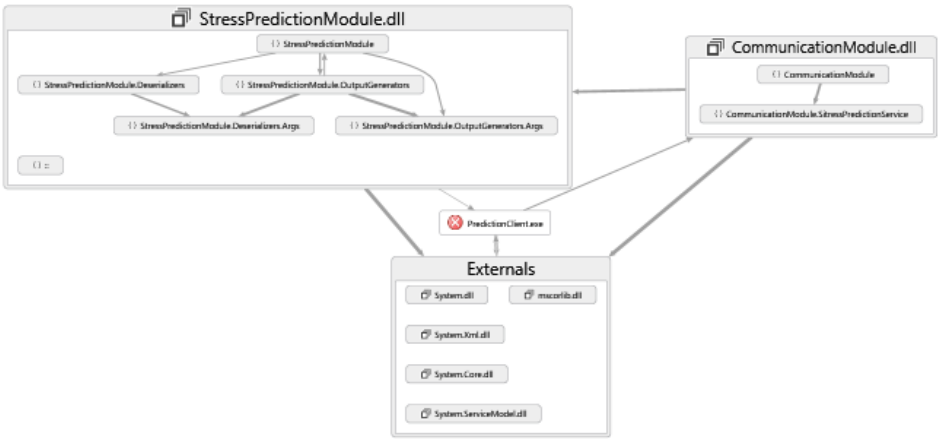


Figura 5.8 *Diagrama pentru partea de client implementată în C#*

Clientul aplicației este implementat în C#. Partea de client e structurată pe 3 module (mai există un modul și pentru partea de silabificare, dar pentru lucrarea de față nu reprezintă un interes în descrierea acestuia), astfel:

- StressPrediction Module – gestionează rezultatele obținute de server și primite prin serviciul web. Pe baza predicților – claselor obținute pentru fiecare instanță – se reface cuvântul cu accentul poziționat
- CommunicationModule – implementează tot ce ține de servicii web și comunicare dintre client-server.
- PredictionClient – implementează interfața grafică de unde utilizatorul gestionează întreaga aplicație.

Figura 5.9 ilustrează diagrama de deployment a aplicației. Aplicația este structurată în două componente: client și server. Clientul este implementat în C# iar server-ul în Java. Comunicarea dintre ele se face prin servicile SOAP. Server-ul este încărcat în container-ul de Glassfish și poate fi accesat la URL : http://localhost:8080/SilabificationPredictor/

Clientul trimite către server mesaje SOAP cu trăsăturile extrase (vectorul de trăsături) în timp ce server-ul primește mesajul, folosește vectorul pentru a construi fișierul arff, antrenează și/sau clasifică și trimite rezultatul înapoi tot prin SOAP, clientului. Acesta procesează predicțile primite și reface cuvântul cu accentul poziționat pe care îl trimite utilizatorului prin interfata grafică.

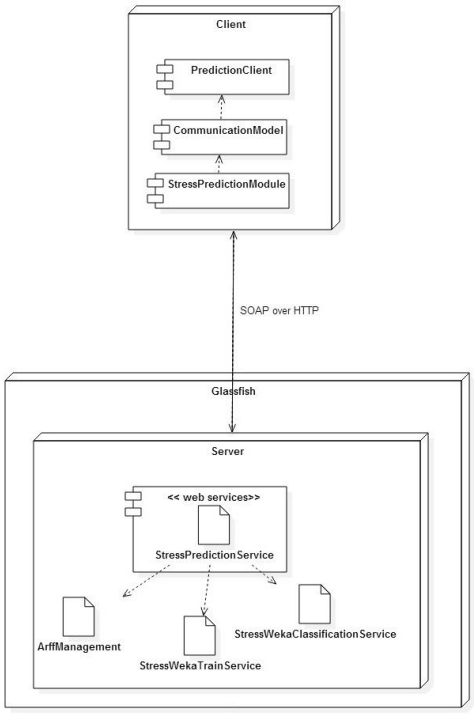


Figura 5.9 *Diagrama de deployment*

Pentru a evidenția mai bine procesul între cele două componente client și server, figura 5.10 arată o diagramă secvențială cu principalele clase și schimbul de mesaje dintre acestea, pe cazul clasificării.

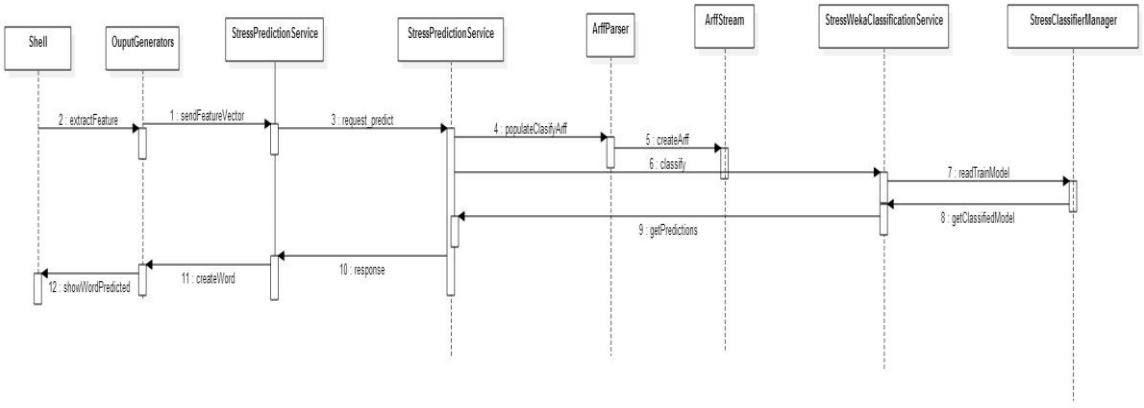


Figura 5.10 *Diagrama de secvență*

5.3. Descrierea implementării

Din punct de vedere al implementării, cea mai importantă parte o reprezintă pachetul de Java care conține întregul mecanism de construire și utilizare a fișierelor arff, antrenare și clasificare. Un punct de interes îl reprezintă și extragerea trăsăturilor și procesarea rezultatelor, a predicților efectuate în urma clasificării și transmise pachetului de C# prin servicii SOA.

1. **Fișierele arff.**

   Așa cum am descris și în capitolul anterior, un fișier de tip arff trebuie să respecte un anumit format. Implementarea este descrisă în figura 5.11. Am construit trei metode care setează relația, atributele și datele. Valorile care completează câmpurile sunt trăsături care au trecut deja prin procesul de extragere a caracteristicilor.

```
public void writeHeader(String name) throws IOException
{
    writer.append(String.format("@relation '%s'", name));
    writer.append("\n");
}

public void writeAttribute(String attributeName, String[] values) throws IOException
{
    String val = flatten(values);
    writer.append(String.format("@attribute %s {%s}", attributeName, val));
    writer.append("\n");
}

public void writeData(String[][] data) throws IOException
{
    writer.append("@data");
    writer.append("\n");
    for (String[] d : data)
    {
        String val = flatten(d);
        writer.append(val);
        writer.append("\n");
    }
}
```

Figura 5.11 *Secvență de cod pentru construirea fișierelor arff.*

2. **Antrenarea și clasificarea**

Codul Java corespunzător procesului de antrenare este ilustrat în figura 5.12. Metoda are ca și parametrii un fișier arff și un id (id-ul e pentru a determina sesiunea, respectiv modelul de antrenare existent și salvat pe disk – exemplul din figură nu utilizează id-ul pentru că face antrenarea unui model nou). Se citește fișierul arff, se scot instanțele care vor fi antrenate cu clasificatorul setat din fișier și se setează indexul clasei pentru ca algoritmul să știe clasa pe care trebuie să o prezică. Următorul pas e de a aplica clasificatorul pe setul de antrenare pregătit anterior.

```
public SilaClassifier Train(int sessionId, ArffSilaStream stream)
        throws DataAccessFault, ClassificationException
{
    Instances trainingSet;

    try (StringReader reader = new StringReader(stream.getContent()))
    {
        trainingSet = new Instances(reader);

        SilaClassifier classifier = new SilaClassifier();
        trainingSet.setClassIndex(trainingSet.numAttributes()-1);
        try {
            classifier.buildClassifier(trainingSet);
            //classifier.finalizeAggregation();
        }
        catch (Exception ex) {
            throw new ClassificationException(ex.getMessage());
        }
        finally
        {
            trainingSet.clear();
            reader.close();
            //fileReader.close();
        }
        return classifier;
    }
    catch (IOException ex) {
        throw new DataAccessFault("Could not access file during read");
    }

}
```

Figura 5.12 *Secvență de cod pentru procesul de antrenare.*

```
public ArrayList<Boolean> Classify(SilaClassifier classifier, ArffSilaStream strea
        throws ClassificationException, DataAccessFault
{
    ArrayList<Boolean> result = new ArrayList<Boolean>();
    try (StringReader reader = new StringReader(stream.getContent()))
    {
        Instances testSet = new Instances(reader);

        testSet.setClassIndex(testSet.numAttributes()-1);
        try {
            for (Instance testEntry : testSet)
            {
                double returnedClass = classifier.classifyInstance(testEntry);
                result.add(returnedClass != 0.0 ? true : false);
            }
        }
        catch (Exception ex) {
            throw new ClassificationException(ex.getMessage());
        }
        finally
        {
            testSet.clear();
            reader.close();
        }
    }
    catch (IOException ex) {
        throw new DataAccessFault("Could not access file during read");
    }
    return result;
}
```

Figura 5.12 *Secvență de cod pentru procesul de clasificare.*

Procesul de clasificare merge pe același mecanism cu cel de antrenare, diferența făcându-se în momentul apelării. Figura 5.13 ilustrează acest aspect.

```
@WebMethod(operationName = "train")
public void train(@WebParam(name = "trainingArg") StressArg[] trainingArg,
        @WebParam(name = "sessionId") int sessionId)
        throws ClassifierNotFoundFault, DataAccessFault, ClassificationException {
    if (!classifierDbManager.ClassifierExists(sessionId))
    {
        ArffStream stream = arffParser.PopulateTrainingArffStream(sessionId, trainingArg);
        Classifier classifier = trainingService.Train(sessionId, stream);
        classifierDbManager.SaveClassifier(classifier, sessionId);
    }
}

@WebMethod(operationName = "predict")
public ArrayList<Boolean> predict(@WebParam(name = "predictArg") StressArg classifyArg, int sessionId)
        throws ClassifierNotFoundFault, DataAccessFault, ClassificationException
{
    ArffStream stream = arffParser.PopulateClassifyArffStream(sessionId, classifyArg);
    return classificationService.Classify(classifierDbManager.ReadClassifier(sessionId), stream);
}
```

Figura 5.13 *Secvență de cod pentru apelarea metodelor de antrenare și clasificare în web service, pe partea de server.*

Diferența constă în utilizarea modelului antrenat de către procesul de predicție și returnarea acestora către client.

3. Servicii Web

Așa cum am menționat și în secțiunile anterioare, partea de client este implementată în C#. Figura 5.14 ilustrează setarea comunicării dintre cele două structuri.

```
  <client>
      <endpoint address="http://localhost:8080/Predictor/PredictionService"
          binding="basicHttpBinding" bindingConfiguration="StressPredictionServicePortBinding"
          contract="PredictionService.StressPredictionService"
          name="StressPredictionServicePort" />
  </client>
system.serviceModel>
```

Figura 5.14 *Secvență de cod pentru setarea endpoint-ului în client, C#*

5.4. Tehnologii utilizate

5.4.1. Weka

Weka este unul dintre cele mai populare echipamente pentru colecțile de algoritmi de ML pentru procesarea datelor. Algoritmii integrați pot fi aplicați direct pe seturile de date prin interfața pe care Weka o oferă sau pot fi apelați de propria aplicație în cod Java. Weka conține instrumente pentru pre-procesarea datelor, clasificare, regresie, clusterizare, asociare de reguli și vizualizare.

Librăria pe care am utilizat-o este 3.5.10 core.weka și are suport doar pentru Java. În timpul testării, lucrând cu seturi mari de date am observat o eroare de Weka, indiferent dacă se utilizează librăria sau interfață. Weka nu face niciodată *garbage collection* după antrenare, iar dacă se încarcă mai multe fișiere arff, acestea rămân într-o coadă. Acest aspect aduce 500 Mb în plus, care rămân în memorie.

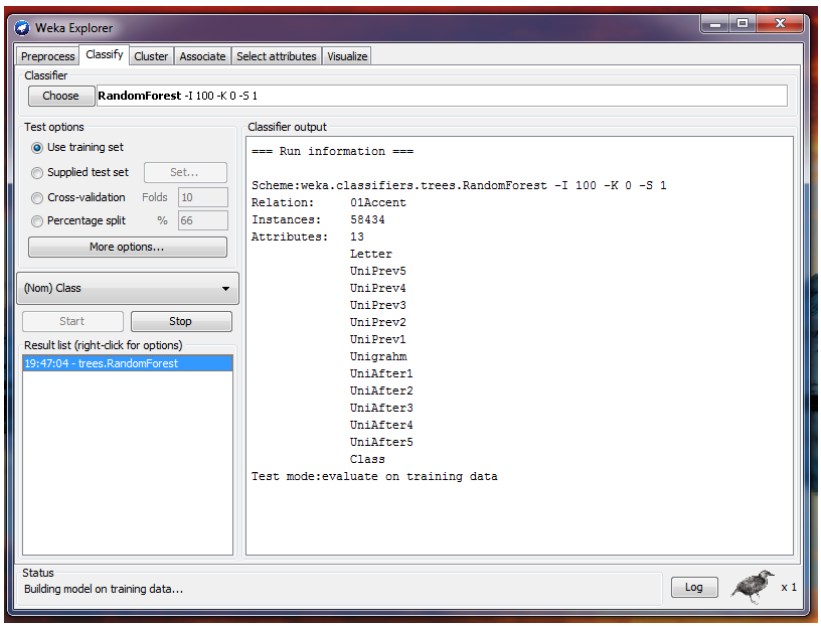


Figura 5.15 *Interfața Weka.*

5.4.2. Servicii Web

Un serviciu Web este o aplicatie Web de tip client-server, în care un server furnizor de servicii (numit si "Service Endpoint") este accesibil unor aplicatii client pe baza adresei URL a serviciului. Serviciul Web si clientii săi pot rula pe platforme diferite si pot fi scrise în limbaje diferite precum aplicația de față deoarece comunicarea se face prin protocoalele standard HTTP, XML, SOAP, JSON, etc. Unul dintre avantajele serviciilor Web este acela că asigură interoperabilitatea unor aplicatii software implementate pe platforme diferite și cu instrumente (“framework”-uri) diferite. Un alt avantaj este dat de principiul *loosely coupled*: mesajele transmise sunt standard și oricare dintre aplicații nu presupune existența la celălalt capăt a altor facilități decât cele conținute în standarde.

În servicile de tip SOAP (*Simple Object Access Protocol*) pe care le implementează și aplicația, cererile și răspunsurile au forma unor documente XML cu un format specific transmise tot peste HTTP. În astfel de servicii furnizorul expune și o descriere a interfetei API sub forma unui document WSDL (Web Service Description Language),tot sub forma de XML, prelucrat de client. Un client trebuie să cunoască metodele oferite de către "Service Endpoint", pe care le poate afla din descrierea WSDL.

Serviciile de tip SOAP oferă mai multă flexibilitate, o mai bună calitate a serviciilor și interoperabilitate decât serviciile REST.

5.4.3. DevExpress

DevExpress este un plugin pentru Visual Studio care vine în ajutorul programatorilor pentru realizarea interfețelor grafice.

Ultima versiune este DevExpress 15.1 și poate fi descărcată de la adresa: https://www.devexpress.com/Home/try.xml

5.4.4. Apache Maven

Apache Maven este un sistem de build și management al proiectelor scris și utilizat în Java. Dintre funcționalități, amintesc descrierea proccesului de build al softaware-ului și descrierea dependențelor acestuia. Practic un fișier .xml descrie toate dependențele necesare proiectului, în timp ce Maven descarcă autoamat .jar-urile necesare. Ultima versiune este Maven 3.3.3 și poate fi descărcată de la adresa: https://maven.apache.org/download.cgi

5.4.5. Glassfish

GlassFish este o aplicație Java de tip server ce permite generarea tehnologiilor de tip enterprise, scalabile ca și servicii adiționale. GlassFish l-am utilizat în contextul creării și utilizării servicilor web ca și Web Server al war-ului generat.

6. Testare şi Evaluare

Testarea, evaluările finale și validarea sistemului este efectuată prin utilizarea tool-ului WEKA şi a seturilor de date obținute din RoSyllabiDict. Datorită numărului mare de cuvinte din dicționar, am putut crea seturi diferite și pentru învățare și pentru testare. Așa cum am menționat în capitolele trecute, WEKA poate primi ca și intrare atât fișiere cu extensia .csv, cât și .arff. Din exeperiența noastră, deocamdată fișierele .arff sunt procesate mai bine de către WEKA în sensul că nu apar erori în încărcarea și citirea fișierelor, nu au nevoie de transformări precum fișierele .csv care trebuie parsate în formatul dat de .arff, etc. Pe de altă parte, pentru a putea lucra la capacitate maximă am fost nevoită să cresc dimensiunea memoriei la 8 GB (maxheap=8192M). Acest lucru ne-a permis să încarcăm fișiere de antrenare și să evaluăm fișierele de testare cu vectori de trăsături pe modelul bigramelor de până la 150,000 de instanțe, iar pentru unigrame până la 100,000 de instanțe.

Din punct de vedere al cuvintelor din RoSyllabiDict, putem crea două tipuri de fișiere: care conțin cuvinte cu diacritice, respectiv care conțin cuvinte fără diacritice.
Testarea am efectuat-o pe diferiți algoritmi de Machine Learning, prezentați în capitolul 3. Tabelul 6.1 face o evaluare generală asupra mai multor algoritmi antrenați pe modele diferite și testați cu același test set. Tabelul 6.1 b) descrie în detaliu conținutul seturilor de date (care diferă din punct de vedere al cuvintelor). Pentru început, antrenarea, testarea și evaluarea am efectuat-o având la bază modelulul vectorului de trăsături fără caracteristici primite de la modulul de silabificare, detaliat în capitolul 4.

Model	**Classifier**	**Correct Classification**	**Precision**	**FP Rate**
Set01	Random Forest	96.37 %	96,4 %	10.3 %
	SMO	93.85%	93.7%	15.2%
	KStar	93.93 %	93.8 %	14.8 %
	Naive Bayes	88.82%	89.4%	16.5%
	Ada Boost	86.33%	86.3%	41.1%
Set02	Random Forest	96.37 %	96,4 %	9.9 %
	SMO	92.82%	92.7%	17.6%
	KStar	93.72 %	93.6 %	14.9 %
	Naive Bayes	88.26%	89.1%	16.1%
	Ada Boost	86.09%	86.2%	42.7%
Set03	Random Forest	96.26 %	96,3 %	10,7 %
	SMO	92.98%	92.9%	17.4%
	KStar	94.09 %	94.0 %	14.9 %
	Naive Bayes	88.12%	89.0%	16.3%

	Ada Boost	86.52%	86.6%	41.1%
Set04	Random Forest	96.58 %	96.6 %	9.4 %
	SMO	92.82%	92.7%	17.6%
	KStar	94.22 %	94.1 %	14.1 %
	Naive Bayes	88.58%	89.3%	16.0%
	Ada Boost	86.5%	86.6%	41.1%
Set05	Random Forest	93.53 %	93.6 %	18.9 %
	SMO	91.54%	91.4%	22.7%
	KStar	92.53 %	92.4%	18.8 %
	Naive Bayes	85.56%	86.3%	22.1%
	Ada Boost	83.92%	83.2%	46.5%

Tabel 6.1 a) *Evaluare pe 5 modele diferite cu același test set și diferiți algoritmi*

Model	Număr de instanțe	Număr de cuvinte
Set 01	58,434	13,210
Set 02	58,626	13,209
Set 03	58,207	13,210
Set 04	58,293	13,211
Set 05	58,288	13,210
*TestSet	3780	860

Tabel 6.1 *b) Mărimea seturilor de date și a test setului*

Conform rezultatelor de clasificare din tabelul de mai sus și a evaluărilor pe baza lui, am ales algoritmul Random Forest pentru a merge mai departe. În continuare sunt prezentate alte teste și evaluări efectuate, precum și analiza acestora.

Cazul 1: Antrenare cu diacritice, evaluare cu diacritice

Tabelul 6.2 schițează rezultatele evaluării aceluiași set de test pe modele diferite de antrenare. Setul de test conține 860 de cuvinte, aproximativ 3780 de instanțe. Toate seturile de date diferă din punct de vedere al cuvintelor utilizate. Primele rezultate sunt prezentate la nivel de instanță.

Run	Instances	Words	Correct Classification	Precision	FP Rate
1	58.434	13.210	96,37 %	0.964	0.103
2	58.626	13.209	96,37 %	0.964	0.099
3	58.207	13.210	96,26 %	0.963	0.107
4	58.293	13.211	96,58 %	0.966	0.094
5	58.288	13.210	93,53 %	0.936	0.189

Tabel 6.2 *Model de antrenare cu diacritice, evaluare cu diacritice. Rezultate la nivel de instanță*

Pentru evaluarea la nivel de cuvânt efectuate pe aceleași seturi de date, acuratețea are o medie de 84,97 %. Acestea sunt prezentate în figura următoare.

Run	Instances	Words	Word accuracy
1	58.434	13.210	87,79 %
2	58.626	13.209	86,74 %
3	58.207	13.210	86,04 %
4	58.293	13.211	86,74 %
5	58.288	13.210	77,56 %

Tabel 6.2 *Model de antrenare cu diacritice, evaluare cu diacritice. Rezultate la nivel de cuvânt*

Cazul 2: Antrenare cu diacritice, evaluare fără diacritice

Tabelul 6.3 schițează rezultatele evaluării aceluiași set de test pe modele diferite de antrenare. Setul de test conține 860 de cuvinte, aproximativ 3780 de instanțe. Toate seturile de date diferă din punct de vedere al cuvintelor utilizate. Rezultate sunt prezentate la nivel de instanță.

Model	Instances	Words	Correct Classification	Precision	FP Rate
Set 01	58,434	13,210	94.27 %	94.3 %	16.5 %
Set 02	58,626	13,209	94.25%	94.3%	16.8 %
Set 03	58,207	13,210	93.61 %	93.6%	17.9%
Set 04	58,293	13,211	94.62 %	94.6%	15.4%
Set 05	58,288	13,210	92.13 %	92.2%	22.9%

Tabel 6.3 *Model de antrenare cu diacritice, evaluare fără diacritice. Rezultate la nivel de instanță*

Cazul 3: Antrenare fără diacritice, evaluare cu diacritice

Tabelul 6.4 schițează rezultatele evaluării aceluiași set de test pe modele diferite de antrenare. Setul de test conține 860 de cuvinte, aproximativ 3780 de instanțe. Toate seturile de date diferă din punct de vedere al cuvintelor utilizate. Rezultate sunt prezentate la nivel de instanță.

Model	Instances	Words	Correct Classification	Precision	FP Rate
Set 01	58,434	13,210	94.94%	95%	15.1%
Set 02	58,626	13,209	94.70%	94.7%	15.1%
Set 03	58,207	13,210	94.78%	94.8%	15.1%
Set 04	58,293	13,211	94.96%	95%	14.7%
Set 05	58,288	13,210	92.29%	92.4%	22.6%

Tabel 6.4 *Model de antrenare fără diacritice, evaluare cu diacritice. Rezultate la nivel de instanță*

Cazul 4: Antrenare fără diacritice, evaluare fără diacritice

Tabelul 6.5 schițează rezultatele evaluării aceluiași set de test pe modele diferite de antrenare. Setul de test conține 860 de cuvinte, aproximativ 3780 de instanțe. Toate seturile de date diferă din punct de vedere al cuvintelor utilizate. Rezultate sunt prezentate la nivel de instanță.

Model	Instances	Words	Correct Classification	Precision	FP Rate
Set 01	58,434	13,210	96.23 %	96.2 %	0.98%
Set 02	58,626	13,209	96.01 %	96.1 %	0.96%
Set 03	58,207	13,210	96.23 %	96.2 %	0.99 %
Set 04	58,293	13,211	96.53 %	96.5 %	0.92 %
Set 05	58,288	13,210	93.35 %	93.4 %	18.9%

Tabel 6.5 *Model de antrenare fără diacritice, evaluare fără diacritice. Rezultate la nivel de instanță*

Cele mai bune rezultate le-am obținut atunci când utilizam aceleași tipuri de fișiere: cu diacritice și modelul de antrenare și setul de evaluare, respectiv fără diacritice și modelul de antrenare și setul de evaluare. Cele mai mici rezultate le-am obținut pentru cazul antrenării cu diacritice și evaluării fără diacritice, având o medie de 93,78%. La prima vedere, cuvintele fără diacritice sunt confuze pe modelul de învățare. Rezultatele evaluăruii sunt prezentate la nivel de instanță. Tabelul 6.6 prezintă media și deviația standard pentru fiecare caz, urmate de graficele aferente.

Caz	Media	Deviația standard
1	95.82	1.28
2	93.78	0.009
3	94.33	0.011
4	95.67	0.013

Tabel 6.6 *Media și deviația standard pentru cele 4 cazuri.*

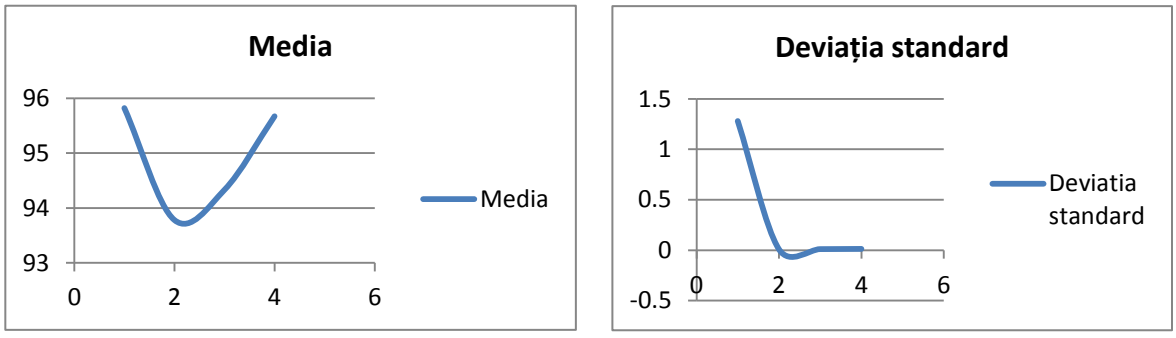


Figura 6.1 *Media și deviația standard pentru cele 4 cazuri.*

O altă evaluare am efectuat-o pentru seturi de date de antrenare din ce în ce mai mari. Tabelul următor prezintă rezultatele atât la nivel de instanță cât și la nivel de cuvânt.

Run	Instances	Words	Correct Classification	Precision	FP Rate	Word Accuracy
1	4.860	1.100	96,04 %	0.96	0.104	84,91 %
2	9.720	2.200	95,72 %	0.95	0.116	83,45 %
3	14.410	3.300	96,12 %	0.96	0.103	87,67 %

Tabel 6.6 *Model de antrenare cu diacritice, evaluare cu diacritice. Rezultate la nivel de cuvânt și instanță pentru seturi de test mari*

Figura următoare ilustrează variația acurateței obținute la nivel instanță și cuvânt utilizînd algoritmul RF și seturi de date de antrenare cu un număr crescător de cuvinte. Putem observa că de la un număr de 3300 până la 13,210 de cuvinte acuratețea la nivel de instanță nu crește considerabil, aceasta fiind de 0,25 procente: de la 96,12 % până la 96,37%. De asemenea, și la nivel de cuvânt creșterea în procente nu e mare : de la 87,79 % până la 87,67 %.

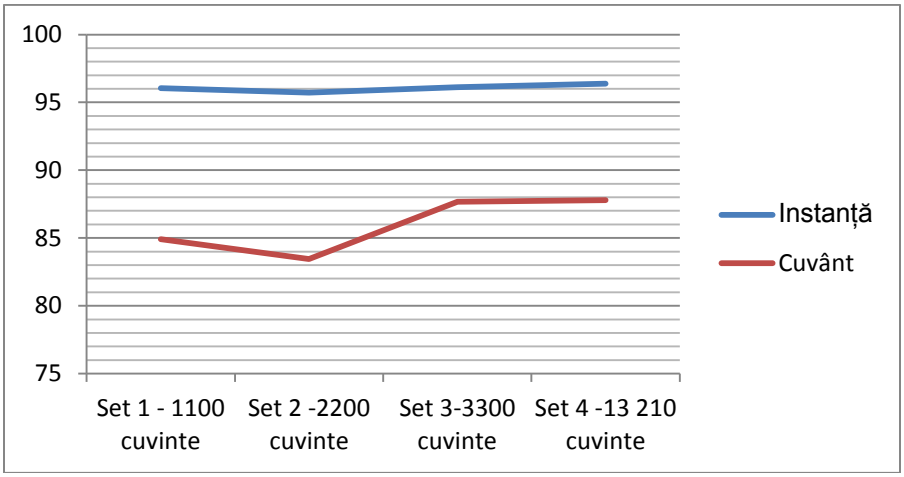


Figura 6.1 *Acuratețea Random Forest pentru identificare accentului*

Pentru primele trei rezultate de mai sus am calculat și numărul de cuvinte pentru care poziția accentului nu a fost identificată, adică modelul nu a prezis nici o poziție pentru accent. Valorile se pot observa în tabelul de mai jos.

Run	Instances	Words	Stress not identified
1	4.860	1.100	143
2	9.720	2.200	318
3	14.410	3.300	389

Tabel 6.7. *Statistica privind cuvintele pentru care nu se prezice nici o poziție a accentului*

Pentru a evidenția acest aspect, am efectuat alte teste pe cuvinte cu diacritice (atât modelul cât și setul de test conțin diacritice). Modelul antrenat cu RF are 4295 de cuvinte (42,665 instanțe) în timp ce setul de test conține 860 de cuvinte din care 78 au fost clasificate greșit. Tabelul 6.8 arată rezultatele de clasificare corectă la nivel de instanță și cuvânt, iar tabelul 6.9 evidențiază câteva din cuvintele pentru care predicția se face greșit.

Run	Instances	Words	Correctly Classified
1	42667	4295	97,45 % – nivel instanță
1	42667	4295	91,75 % - nivel cuvânt

Tabel 6.8. *Statistica privind clasificarea cuvintelor cu diacritice la nivel instanță și cuvânt*

Nr	Actual	Predicție
1.	Abații	Abații
2.	Algei	Algei
3.	Astrofizicii	Astrofizicii
4.	Cronometra	Cronometra
5.	Deasupra	Deasupra
6.	Ecologii	Ecologii
7.	Petrecerea	Petrecerea
8.	Piftie	Piftie
9.	Oracoleler	Oracolelor
10.	Liberi	Liberi

Tabel 6.9. *Exemple cuvinte clasificate incorect pentru testele descrie în tabelul 6.8*

Cazul bigramelor nu ne avantajează. Rezultatele obținute nu aduc îmbunătățiri, ba mai mult decât atât îngreunează procesul de clasificare prin creșterea timpului de clasificare. Rezultatele obținute sunt sub 94%, ceea ce ne-a determinat să alegem varianta cu unigrame.

Totodată trebuie să menționez că testând aceste cazuri, am observat că după finalizarea procesului, nu se face garbage collection așa cum ar fi normal. Deși este implementat acest mecanism și poate fi executat și explicit, eroarea persistă. Astfel, modele rămân în memorie și adăugă aproximativ câte 500Mb în JVM, până când se închide aplicația sau interfața pusă la dispoziție de Weka.

Am calculat și eroarea pe train set, iar rezultatele sunt prezentate îm tabelul următor:

Model	Instances	Words	Correct Classification	Precision	FP Rate
Set 01	58,434	13,210	99.97 %	1 %	0.1%
Set 02	58,626	13,209	99.97 %	1 %	0.1%
Set 03	58,207	13,210	99.97 %	1 %	0 %
Set 04	58,293	13,211	99.97 %	1 %	0.1%
Set 05	58,288	13,210	99.97 %	1 %	0%

Tabel 6.10 *Evaluarea pe train set*

Evaluarea testelor efectuate pe baza vectorului de trăsături cu două trăsături primite prin procesul de silabificare:

Tabelul 6.11 ilustrează rezultatele efectuate pe baza celui de-al doilea tip de vector de trăsături, prezentat în capitolul 4. Setul de test conține 860 de cuvinte, aproximativ 3780 de instanțe. Toate seturile de date diferă din punct de vedere al cuvintelor utilizate. Rezultate sunt prezentate la nivel de instanță.

Model	Instances	Words	Correct Classification	Precision	FP Rate
Set 01	58,434	13,210	95.55 %	95.6 %	12.5%
Set 02	58,626	13,209	94.97 %	94.1 %	11.3%
Set 03	58,207	13,210	95.42 %	95.1 %	7.3 %
Set 04	58,293	13,211	95.8 %	95.0 %	12.1%
Set 05	58,288	13,210	93.70 %	93.2%	14.5%

Tabel 6.11 *Evaluarea pentru clasificarea cu cel de-al doilea tip de vector*

Rezultatele obținute utilizând trăsături de la modulul de silabificare nu ilustrează vreo îmbunătățire în clasificare. Se poate însă analiza cazurile care eșuează și revizui vectorul de trăsături ca o dezvoltare ulterioare.

7. Manual de Instalare si Utilizare

Aplicația oferă o soluție software pentru utilizatorii care doresc să o folosească pentru a obține identificarea accentului sau despărțirea în silabe a cuvintelor, după caz. De asemenea, aplicația poate fi integrată cu altele mai complexe în vederea realizării unui sistem multifuncțional utilizat la scară mare.

a. Instalarea aplicației

i. Cerințe hardware

Pentru ca aplicația software să ruleze în cele mai bune condiții sunt necesare câteva cerințe hardware:

- Cerințe minime:
  - Intel Core i3 CPU cu o frecvență de cel puțin 2Ghz
  - 4GB RAM (sau chiar mai mult, depinde de mărimea seturilor de date utilizate pentru antrenarea și testarea sistemului)
- Cerințe recomandate:
  - Intel Core i5 CPU cu o frecvență de 2,50 Ghz
  - 8GB RAM

ii. Cerințe software

Din punct de vedere al cerințelor software, avem:

- Java SE Runtime Environment (JRE) 1.7+ (open-source via http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html)

b. Utilizarea aplicației

Scopul sistemului este de a oferi utilizatorilor o aplicație care să le permită să facă procesare pe texte în limba română. Aplicația este ușor de utilizat, are o interfață simplă ce nu le permite celor ce o folosesc să se piardă. Interfața pentru utilizator este prezentată în figura 7.1.

Aplicația permite utilizatorilor următoarele funcționalități:

- Încărcarea de fișiere text care conțin texte în limba română. Pentru ca viteza de procesare să fie bună, se recomandă un fișier de până la 15,000 de cuvinte. Fișierele sunt acceptate sub forma .txt. Sistemul se ocupă mai departe de alte prelucrări și transformări.
- Încărcarea unui cuvânt. Cuvântul se poate scrie direct cu diacritice. Sistemul se ocupă mai departe de alte prelucrări și transformări.
- Alegerea opțiunii de operație: identificarea accentului sau despărțirea în silabe.

- Vizualizarea rezultatelor obținute: textul dat ca intrare despărțit în silabe sau cu accentul poziționat pentru fiecare cuvânt. Fișierul obținut la ieșire este tot de tipul .txt sau cuvântul cu accentul prezis.

În continuare voi prezenta pașii de urmat pentru cazul încărcării unui cuvânt. Interfața aplicației este simplă și orice tip de utilizator poate să o acceseze.

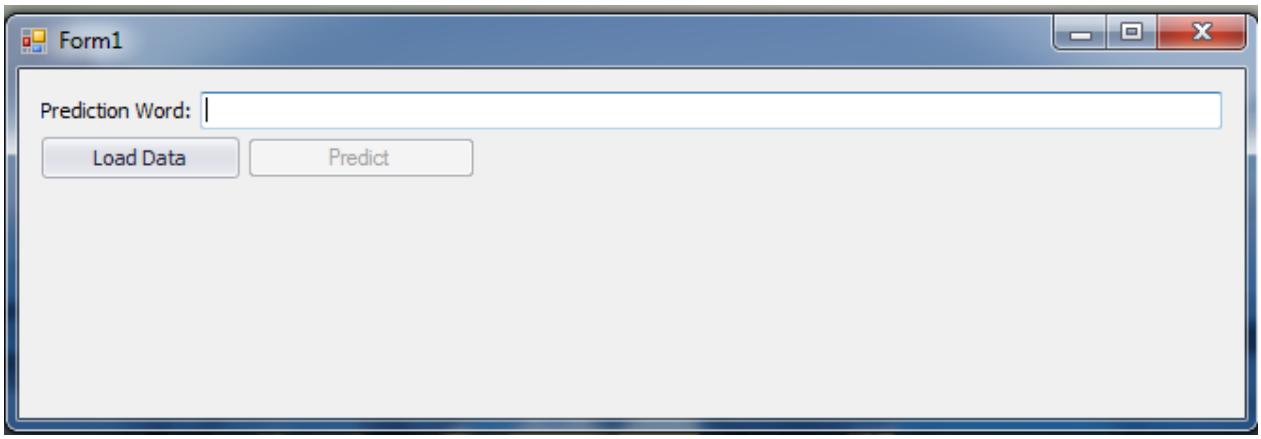


Figura 7.1. *Interfața aplicației*

Primul pas după afișarea ferestrei e de a selecta butonul de *LoadData.* În acest moment se pregătește modelul de antrenare pentru ca aplicația să fie capabilă să prezică accentul.

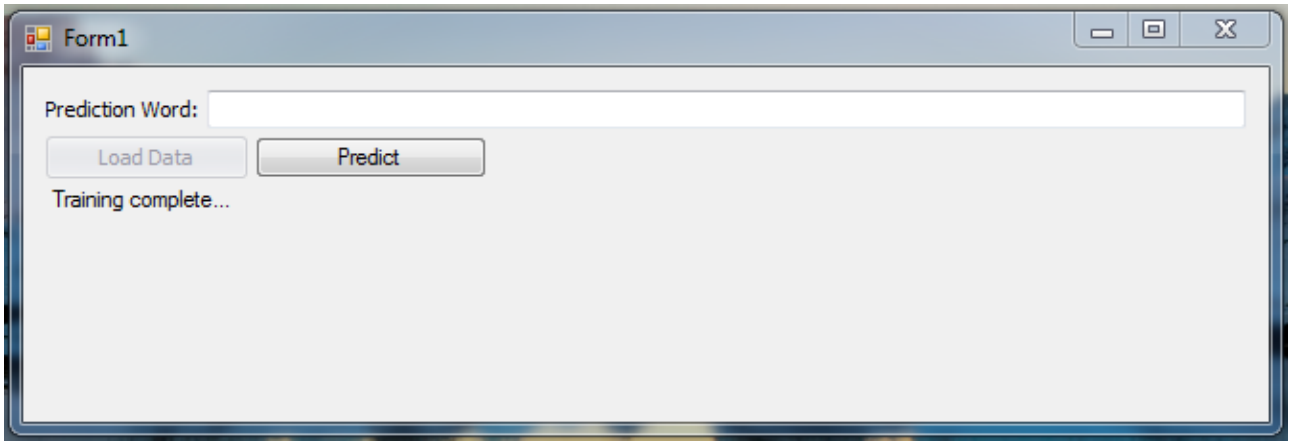


Figura 7.2. *Interfața aplicației după încărcarea modelulului de antrenare*

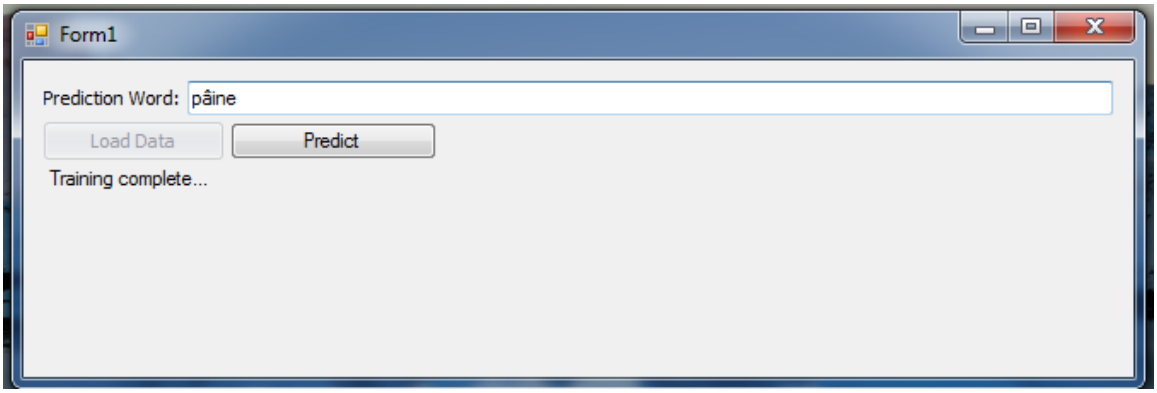


Figura 7.3. *Interfața aplicației după introducerea cuvântului*

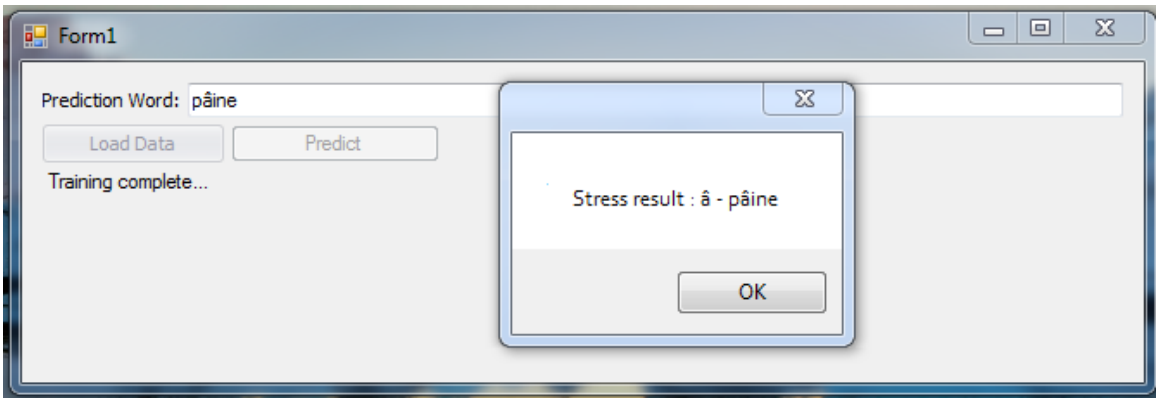


Figura 7.4. *Interfața aplicației după prezicerea accentului*

Aplicația integrează și modulul de silabificare, astfel că cele două operații se vor face de pe ce aceași interfață grafică. Un exemplu al interfeței este prezentat în figura următoare. De menționat că operațile nu se pot executa deodată. Utilizatorul trebuie să aștepte până ce prima operație se va finaliza pentru a o selecta pe cealaltă.

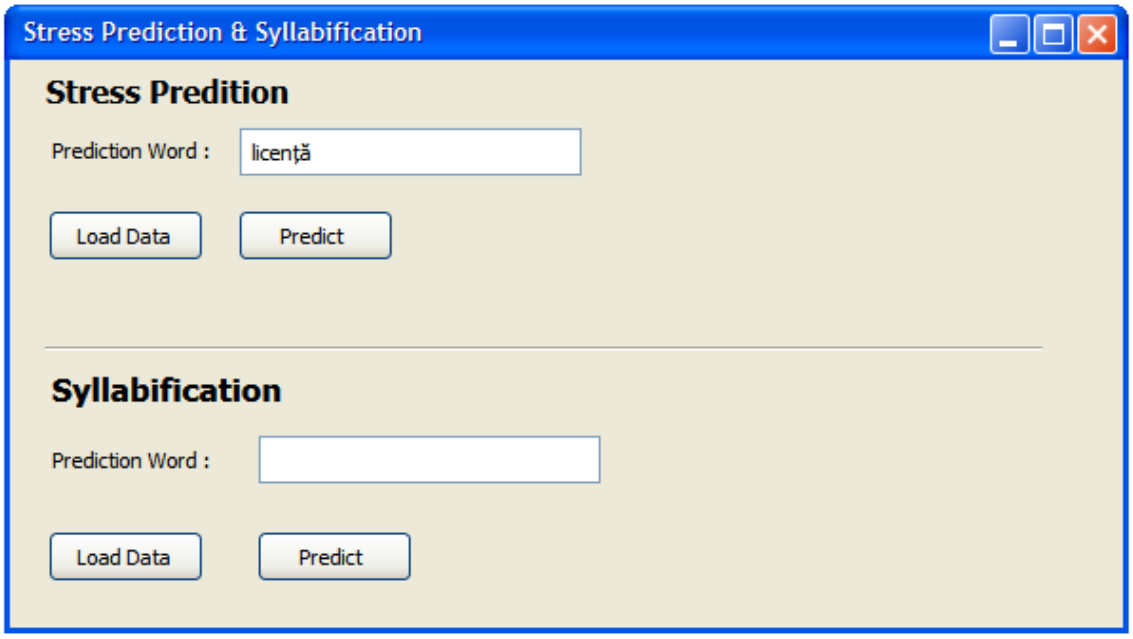


Figura 7.5. *Interfața aplicației după prezicerea accentului*

8. Concluzii

Aplicația a fost dezvoltată și implementată pe baza algoritmilor de învățare supervizată, ceea ce presupune o atentă analiză asupra trăsăturilor pentru crearea vectorului de trăsături și comportamentul acestuia în învățare.

a. Analiză critică a rezultatelor obținute

Scopul și obiectivele proiectului identificate și descrise în capitolul 2 al acestei lucrări, au fost îndeplinite. Sistemul de identificare al accentului, integrat cu cel de despărțire în silabe este unul funcțional, rezultatele obținute fiind pozitive. Seturile de date obținute pe baza dicționarului (520,000 de cuvinte) sunt consistente, iar testele efectuate au relevat faptul că pentru sistemul nostru sunt într-adevăr relevante raportate la cerințele utilizatorului. Seturile de date utilizate ca și seturi de antrenare sunt salvate si utilizate și în alte clasificări. Un model bun este capabil să prezică poziția accentului pentru orice set de test.

Totodată, utilizatorilor le este permis să încarce fișiere considerate de către sistem seturi de test, astfel încât să primească rezultatele pentru alte noi cuvinte fie ele și diferite față de ce pune la dispoziție RoSyllabiDict. Sistemul e capabil să prezică pentru orice cuvinte din limba română, el având în spate un model din care învață.

Analizând rezultatele obținute pentru prezicerea accentului, observăm că problema identificării acestuia nu e atât de predictivă și mult mai sensibilă. Accentul în limba română nu se poziționează explicit în scriere, ci doar în vorbire. Totodată, nu există reguli bine definite, clare (sau care ar putea fi extrase din cuvintele limbii române) ce pot fi aplicate ca șabloane. Știm că accentul întotdeauna se pune pe o vocală, dar nu e suficient. Știm și că în marea majoritate a cazurilor, accentul cade pe penultima silabă, dar acestea sunt doar statistici.

Din punct de vedere al cerințelor non-funcționale identificate tot în capitolul 2, acestea au fost în cea mai mare parte atinse. Astfel, cerința de scalabilitate este satisfăcută. Sistemul este capabil să proceseze un număr mare de cuvinte, rezultatele rămânând la fel de consistente. Referindu-mă la performanță, timpul de încărcare al fișierelor e considerabil mic. Timpul de efectuare al clasificării diferă atât din cauza numărului de cuvinte dar și din cauza algoritmului ales pentru a efectua operația. Implementarea acestora diferă - Random Forest are la bază arbori, în timp ce SVM se bazaează pe vectori - și ca urmare și timpul de execuție diferă.

Cerința de portabilitate este la rândul său îndeplinită, implementarea am efectuat-o utilizând tehnologiile de Java și C# integrându-le prin servicii Web. Astfel, aplicația poate fi utilizată de pe orice sistem.

În ceea ce privește utilizabilitatea, sistemul poate fi manipulat cu ușurință. Utilizatorul nu trebuie să aibă decât cunoștințele minime utilizării unui calculator, interfața aplicației fiind foarte sugestivă.

Aplicația este disponibilă 24/7, dar trebuie să rulzez pe un sistem local.

b. Dezvoltări și îmbunătățiri ulterioare

Aplicația, așa cum e finalizată în momentul de față, permite identificarea accentului (și silabificarea cuvintelor) din texte pe limba română. Scopul acesteia și ideea implementării prin algoritmi de învățare face ca sistemul să poată fi integrat cu alte sisteme de procesare a textelor.

O primă dezvoltare ulterioară ar putea fi integrarea în aplicație a mai multor tipuri de algoritmi de învățare care să îi permită utilizatorului de unde alege. De asemenea, s-ar putea implementa și o soluție pentru interfață care să aibă posibilitatea de a oferi o comparație între doi sau mai mulți algoritmi aleși de către utilizator. Apoi, s-ar putea oferi șansa de a alege din interfață tipul vectorului de trăsături și din nou de a se face o comparație între aceștia.

Din punct de vedere al vectorului de trăsături, acesta ar putea trata și problema părții de vorbire a cuvântului – POS (eng. *Part-of-Speech*). Ar fi interesant de văzut ce se întâmplă cu rezultatele de clasificare, dacă cu implementarea de acum media e de 95,82%. Totodată, POS-ul ar putea rezolva și problema omonimelor.

Predicțile obținute utilizând trăsături în urma procesului de silabificare nu au dat rezultate mai bune decât utilizând primul vector. Merit încercată cazul unui vector cu alte tipuri de trăsături primite în urma silabificării. S-ar putea lua în considerare de exemplu, silaba accentuată. Aici însă trebuie atenție asupra acestei caracteristici deoarece în cazul în care despărțirea în silabe nu s-a efectuat corect, eroare de predicție va fi propagată și la procesul de identificare al accentului.

Prin ultimele două îmbunătățiri s-ar putea rezolva și problema cuvintelor pentru care predicția nu identifică vreun accent.

O altă îmbunătățire ar putea fi internaționalizarea sau mai bine spus adaptarea sistemului dat fiind că soluția implementată nu e strâns corelată de limba română și poate fi integrată și la altele.

Bibliografie

[1] Academia Română, DOOM - Dictionarul Ortografic, Ortoepic si Morfologic al Limbii Romane (editia a II-a, revizuita si adaugita), București: Univers Enciclopedic Gold, 2010.

[2] Franklin Mark LIANG, PhD Thesis: *Word Hyphenation by Computer*, Standford University, 1983.

[3] Lorenzo CIONI, *An algorithm for the syllabification of written Italian*, paper accepted at the 5th International Symposium on Social Communication, Santiago de Cuba, Cuba, 1997 (also published on Quaderni del Laboratorio di Linguistica, issue 11, Scuola Normale Superiore)
Disponibil on-line: http://www.di.unipi.it/~lcioni/papers/1997/LC.Sillabatore.pdf

[4] Liviu P. DINU, Vlad NICULAE, Octavia-Maria SULEA, *Romanian Syllabication using Machine Learning*, Proceedings 16th International Conference, TSD 2013, Pilsen, Czech Republic
Disponibil on-line: http://vene.ro/papers/tsd13.pdf

[5] Liviu P. DINU, Vlad NICULAE, Octavia-Maria SULEA, *Romanian Syllabication using Machine Learning*, Proceedings 16th International Conference, TSD 2013, Pilsen, Czech Republic
Disponibil on-line: http://vene.ro/papers/tsd13.pdf

[6] Tiberiu BOROȘ, *A Unified Lexical Processing Framework Based on the Margin Infused Relaxed Algorithm. A Case Study on the Romanian Language*, Proceedings of The 9th Conference RANLP, Hissar, Bulgaria, September 10–13, 2013
Disponibil on-line: http://www.aclweb.org/anthology/R13-1012

[7] John G. Cleary, Leonard E. Trigg: *K*: An Instance-based Learner Using an Entropic Distance Measure.* In: 12th International Conference on Machine Learning, 108-114, 1995.

[8] Kseniya Rogova, Kris Demuynck, Dirk Van Compernolle. 2013. *Automatic syllabification using segmental conditional random fields*. In Computational Linguistics in the Netherlands Journal 3 (2013), 34-48.

[9] Bing LIU, *Sentiment Analysis and Opinion Mining*, Morgan &Claypool Publishers, May 2012.
Franklin Mark LIANG, PhD Thesis: *Word Hyphenation by Computer*, Standford University, 1983.

[10] Liviu P. DINU, *Quantitative, cognitive and computational aspects of the Romanian syllables*, Revue Romaine de Linguistique, LI(3-4), 2006, pag. 477-498
Disponibil on-line: http://www.lingv.ro/RRL%2034%202006%20Liviu%20Dinu.pdf

[11] Liviu P. DINU, *Despărțirea automată în silabe a cuvintelor din limba română. Aplicații în construcția bazei de date a silabelor limbii române*, Raport de cercetare în proiect finanțat de CNCSIS în anul 2004, cod 217AT, comisia 2 (durata: 1 an).

[12] Ana-Maria BARBU, *Romanian Lexical Data Bases: Inflected and Syllabic Forms Dictionaries*, Proceedings of LREC20081, Marrakech, Maroc, 26-31May,2008 Disponibil on-line: http://www.lrecconf.org/proceedings/lrec2008/pdf/495_paper.pdf

[13] Eugeniu OANCEA, Adriana BADULESCU, *Stressed Syllable Determination for Romanian Words within Speech Synthesis Applications*, International Journal of Speech Technology, pag. 237-246, 2002

[14] Susan BARLETT, Grzegorz KONDRAK, Colin CHERRY, *Automatic Syllabification with Structured SVMs for Letter-To-Phoneme Conversion*, Proceedings of Association of Computational Linguistics, 2008
Disponibil On-line: http://www.aclweb.org/anthology/P08-1065

[15] Qing Dou, Shane Bergsma, Sittichai Jiampojamarn, and Grzegorz Kondrak. 2009. *A ranking approach to stress prediction for letter-to-phoneme conversion.* In Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing

[16] Gabriela Pana Dindelegan. 2013. ˇ *The Grammar of Romanian*. Oxford University Press.

[17] Liviu P. DINU, Anca DINU, *On the data base of Romanian syllables and some of its quantitative and cryptographic aspects*, Proceedings 5th LREC 2006, pag. 1795-1799, Genova, Italy, may 2006
Disponibil On-line: **http://aclweb.org/anthology/L06-1179**

[18] William B. Cavnar, John M. Trenkle, *N-Gram-Based Text Categorization*, in Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 1994

[19] Ron Kohavi, *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*, in the International Joint Conference on Artificial Intelligence (IJCAI), 1995

[20] Weka Official Site http://www.cs.waikato.ac.nz/

Anexe

Anexele cuprind două articole științifice a căror co-autor sunt. Acestea dovedesc, de asemenea, și participarea la Sesiunea de Comunicări Științifice ale Studenților din data de 19 iunie 2015 și înscrierea la International Conference on Intelligent Computer Communication and Processing (ICCP 2015).