Aplicații de Machine Learning Folosind CUDA

Proiect

Arhitectura Informației

Student
Bianca-Alexandra GHIORGHIU

Coordonator
George Valentin STOICA

UNIVERSITATEA
POLITEHNICA
DIN BUCUREȘTI

28 mai 2023

Cuprins

Introducere

Descrierea Proiectului

- Scopul principal al acestui proiect este de a realiza o analiză detaliată asupra implementării algoritmilor de machine learning pe arhitecturile GPU și CPU.
- Proiectul urmărește să investigheze impactul metodelor de vectorizare a cuvintelor și algoritmilor de machine learning în clasificarea tweet-urilor din setul de date Hateval-2019, având ca obiectiv identificarea dacă acestea sunt instigatoare la ură sau nu.
- Aplicațiile de machine learning vor fi dezvoltate și implementate pe arhitecturi GPU Tesla T4.
- Se va utiliza biblioteca Nvidia Rapids, în special cuDF și cuML, pentru a beneficia de funcționalitățile de accelerare GPU.
- Proiectul va examina impactul bibliotecilor de accelerare GPU în procesul de încărcare a datelor și performanța algoritmilor de machine learning.
- Se va efectua o comparație între timpii de execuție obținuți pe arhitectura GPU și cea CPU, pentru a evalua în mod obiectiv beneficiile aduse de procesarea pe GPU în contextul machine learning.

Structura

1. **Introducere:** Această secțiune oferă o introducere generală în proiect și prezintă scopul și obiectivele acestuia. De asemenea, evidențiază structura proiectului, indicând cum sunt organizate capitolele și subcapitolele.
2. **Context Teoretic:** Acest capitol oferă o perspectivă teoretică asupra conceptelor și tehnologiilor fundamentale utilizate în proiect. Se discută despre diferitele tipuri de învățare automată, cum ar fi învățarea supervizată, nesupervizată, semi-supervizată și prin recompensă. De asemenea, se abordează programarea GPU și CUDA în contextul aplicațiilor de învățare automată și se prezintă metodele de vectorizare a cuvintelor și algoritmi folosiți.
3. **Descrierea Implementării:** Aici sunt prezentate detalii practice despre implementarea proiectului. Se discută despre setul de date utilizat și se oferă informații despre implementarea propriu-zisă, cum ar fi tehnologiile și bibliotecile folosite. De

asemenea, sunt prezentate etapele implementării, oferind o imagine de ansamblu a procesului de implementare.

4. **Rezultate:**Această secțiune prezintă rezultatele obținute în urma implementării proiectului. Se discută despre rezultatele atinse atât folosind procesarea pe GPU, cât și folosind procesarea pe CPU. De asemenea, se efectuează o comparație între diferitele metode utilizate în ceea ce privește acuratețea, timpul de execuție și utilizarea GPU.
5. **Concluzii:** În acest capitol, se prezintă concluziile proiectului, evidențiind rezultatele și îndeplinirea obiectivelor propuse.

Capitolul 1

Context Teoretic

1.1 Machine Learning

Învățarea automată este o ramură a inteligenței artificiale care se concentrează pe studiul metodelor care permit sistemelor să se îmbunătățească autonom din experiență [1]. Există trei componente fundamentale ale învățării automate:

Setul de date: Eșantioanele de date folosite pentru a antrena algoritmi de învățare automată.

Caracteristici: Informații esențiale, deoarece indică sistemului pe ce trebuie să se concentreze.

Algoritm: Aceeași sarcină poate fi abordată cu algoritmi diferiți. Criteriile privind precizia, calitatea rezultatelor și puterea de procesare depind de algoritmul ales. De asemenea, este de remarcat faptul că, ocazional, combinarea a doi sau mai mulți algoritmi poate îmbunătăți performanța (învățare în ansamblu).

Cele cinci categorii principale de învățare automată sunt învățarea supervizată, învățarea nesupervizată, învățarea semisupervizată, învățarea prin recompensă și reducerea dimensionalității.

1.1.1 Învățarea Supervizată

După analizarea unei cantități mari de date cu etichete corecte și identificarea relațiilor dintre date care conduc la răspunsuri corecte, sistemele de învățare supervizată pot face predicții. Sistemele sunt numite "supervizate" deoarece li se furnizează date cu etichete adecvate. Acestea fac predicții, care sunt corectate de etichetă. Regresia și clasificarea sunt cele două aplicații cele mai importante ale învățării supervizate [2].

Regresia: Se încearcă să se prezică un rezultat continuu prin asocierea variabilelor de intrare cu o funcție continuă.

Clasificarea: Într-o sarcină de clasificare, se încearcă să se prezică rezultate discrete. În esență, variabilele de intrare sunt asociate cu clase discrete. Există două tipuri de clasificare: clasificare binară, care se ocupă de două clase, și clasificare multi-clasă, care se ocupă de mai multe clase.

1.1.2 Învățarea Nesupervizată

Învățarea nesupervizată permite abordarea problemelor unde nu există cunoștințe despre rezultatele așteptate. Prin gruparea datelor în funcție de corelațiile dintre variabile, se poate genera structură din date în care efectele variabilelor sunt neclare. Gruparea diferă de clasificare în sensul că utilizatorul nu specifică categoriile. În învățarea nesupervizată, nu există niciun feedback bazat pe acuratețea predicțiilor [2].

1.1.3 Învățarea Semi-supervizată

Învățarea semi-supervizată se situează între învățarea nesupervizată și învățarea supervizată, având caracteristica principală că unele date nu au etichete. În domeniul învățării automate s-a demonstrat că utilizarea datelor fără etichete împreună cu o mică cantitate de date etichetate poate îmbunătăți semnificativ acuratețea învățării.

1.1.4 Învățarea prin Recompensă

Modelele bazate pe învățarea prin recompensă generează predicții prin primirea recompenselor sau sancțiunilor pentru acțiunile întreprinse într-un domeniu. Un model de învățare prin recompensă creează o politică care specifică abordarea optimă pentru maximizarea recompenselor [3].

1.1.5 Reducerea Dimensionalității

Redimensionarea dimensionalității este procesul de reducere a numărului de caracteristici luate în considerare prin descompunerea acestora într-un set de valori principale. Analiza componentelor principale (PCA) este una dintre cele mai utilizate tehnici pentru redimensionarea dimensionalității; implică transformarea datelor cu dimensionalitate mare într-un spațiu mai mic [4].

1.2 Programarea GPU și CUDA în aplicațiile de învățare automată

1.2.1 Introducere în programarea GPU și CUDA

Programarea GPU implică scrierea de cod care este executat pe GPU. Inițial, GPU-urile au fost concepute pentru redarea graficii în jocurile video, dar potențialul lor de procesare paralelă este acum exploatat pentru o varietate de sarcini de calcul, în special în domeniile învățării automate și în data science.

GPU-urile conțin multe nuclee (sute sau chiar mii) care le permit să gestioneze mai multe sarcini simultan, ceea ce le face potrivite pentru nivelurile ridicate de procesare cerute de algoritmii de învățare automată. Ele sunt utile în special pentru sarcinile care pot fi împărțite în părți mai mici, independente și executate în paralel, cum ar fi operațiile matriciale și alte calcule matematice care sunt utilizate în mod obișnuit în învățarea automată.

CUDA (Compute Unified Device Architecture) este o platformă de calcul paralel și un model API creat de NVIDIA. Aceasta permite dezvoltatorilor să utilizeze o unitate de

procesare grafică cu CUDA pentru procesare de uz general, o abordare denumită GPGPU (General-Purpose computing on Graphics Processing Units). CUDA oferă acces direct la setul de instrucțiuni virtuale și la memoria elementelor de calcul paralel din GPU CUDA.

1.2.2 Programarea GPU și CUDA în Machine Learning

Aplicarea programării CUDA și GPU în învățarea automată devine din ce în ce mai populară din cauza cerințelor de calcul ale algoritmilor moderni de învățare automată. Acești algoritmi implică adesea operații matriceale și vectoriale complexe, care sunt paralelizabile și, prin urmare, pot beneficia în mod semnificativ de avantajele arhitecturale ale GPU-urilor.

Sarcinile de învățare automată, cum ar fi antrenarea rețelelor neuronale, implică efectuarea multor înmulțiri de matrice, o sarcină care poate fi paralelizată și, astfel, executată mai rapid pe un GPU. În schimb, procesoarele tradiționale cu mai puține nuclee ar executa aceste multiplicări de matrice în mod secvențial, ceea ce ar dura mult mai mult timp.

CUDA permite utilizarea eficientă a arhitecturii cu mai multe nuclee a GPU-ului prin intermediul funcțiilor sale de bază, cum ar fi CUDAthreads, warps și blocks. Aceste elemente permit un control avansat asupra execuției sarcinilor pe GPU, ceea duce la aplicații de învățare automată foarte eficiente și scalabile.

1.2.3 Utilizarea bibliotecilor CUDA și a bibliotecilor accelerate de GPU

Există mai multe biblioteci bazate pe CUDA sau accelerate de GPU care simplifică dezvoltarea aplicațiilor de învățare automată pe GPU. Aceste biblioteci oferă interfețe de nivel înalt, ușor de utilizat și sunt optimizate pentru performanță pe GPU-urile NVIDIA. Unele dintre aceste biblioteci includ:

- **cuDNN (biblioteca CUDA Deep Neural Network):** Bibliotecă cu accelerare GPU pentru rețele neuronale profunde. Oferă funții optimizate pentru algoritmi de învățare profundă.
- **TensorRT:** Bibliotecă de optimizare a inferenței de învățare profundă și de execuție de înaltă performanță pentru implementarea în producție a aplicațiilor de învățare profundă.
- **NVIDIA DALI:** Bibliotecă pentru încărcarea și preprocesarea datelor pentru a îmbunătăți performanța pipeline-ului de date pentru învățarea profundă.
- **NVIDIA RAPIDS:** Bibliotecă software open-source concepută pentru a permite rularea pe GPU a unor pipe-line-uri de data science și de învățare automată la scară largă.

Pentru realizarea acestui proiect, s-a folosit NVIDIA RAPIDS, pentru că este o colecție de biblioteci care oferă accelerare GPU pe integul pipe-line al aplicației de învățare automată. Câteva dintre componentele principale sunt:

- **cuDF (CUDA DataFrame):** Bibliotecă pentru DataFrames accelerată GPU pentru încărcarea, unirea, agregarea, filtrarea și manipularea datelor. Oferă un API asemănător cu pandas, diferența esențială fiind că operațiile cuDF sunt efectuate pe GPU, ceea ce duce la creșteri semnificative de viteză.
- **CuML (CUDA Machine Learning):** Suită de biblioteci care implementează algoritmi de învățare automată și funcții având API-uri compatibile cu alte proiecte RAPIDS. cuML permite executarea de sarcini tradiționale de învățare automată pe GPU fără a intra în detaliile programării CUDA.
- **cuGraph (CUDA Graph):** Bibliotecă accelerată GPU pentru analiza grafică, care oferă funcționalități precum traversarea, fluxul, componentele, conectivitatea etc. într-un mod accelerat de GPU. cuGraph își propune să ofere un API asemănător cu NetworkX.
- **cuSpatial (CUDA Spatial):** Bibliotecă accelerată GPU pentru procesarea datelor geospațiale și spațio-temporale. Oferă un API de tip pandas care simplifică procesul de manipulare a datelor spațiale cu operații complexe, cu utilizare intensivă a memoriei.

Bibliotecile RAPIDS sunt concepute pentru a lucra împreună și se integrează perfect cu biblioteci de data science, cum ar fi Dask pentru calcul distribuit, și funcționează bine cu instrumentele PyData, cum ar fi NumPy, Pandas și Scikit-learn.

1.3 Metode de Vectorizare a Cuvintelor

1.3.1 Word2Vec

Word2Vec este o metodă de procesare a limbajului natural care convertește cuvintele în reprezentări numerice cunoscute sub numele de "word embeddings". Aceasta se bazează pe premisa potrivit căreia contextul în care apare un cuvânt poate deduce semnificația acestuia [5].

Există două variante ale modelului Word2Vec [6], așa cum se poate observa în 1.1:

1. **Continuous Bag of Words (CBOW):** Această metodă prezice cuvântul țintă din contextul său. Contextul este definit ca o secțiune de cuvinte din jur.
2. **Skip-Gram:** Această metodă prezice cuvintele de context din cuvântul țintă.

Skip-gram

Word2Vec este o rețea neuronală cu un singur strat. Reprezentările de cuvinte sunt rezultatul antrenamentului acestei rețele. În loc de a utiliza modelul pentru predicție după instruire, ponderile rețelei sunt folosite pentru a reprezenta cuvintele [5].

Intrarea este cuvântul țintă codificat într-un vector one-hot. Acesta este un vector binar de dimensiunea vocabularului, cu 0-uri, cu excepția unui 1 la indexul cuvântului de intrare. Stratul de ieșire este un strat softmax care produce o distribuție de probabilitate asupra tuturor termenilor din vocabular pentru fiecare poziție de context [6].

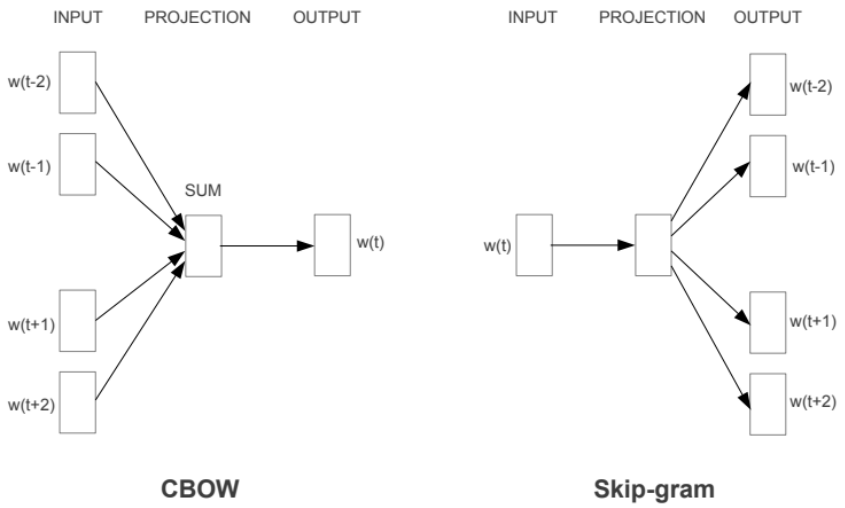


Figura 1.1: Variante word2vec [6]

Straturile de intrare și de ieșire sunt ambele conectate complet la un strat ascuns. Dimensiunea stratului ascuns influențează dimensiunea vectorului de cuvinte (de exemplu, dacă stratul ascuns conține 300 de neuroni, vectorul de cuvinte va avea 300 de dimensiuni).

Funcția care trebuie maximizată pentru modelul skip-gram este [5]:

$$L = \frac{1}{T}\sum_{t=1}^{T}\sum_{-n\leq j\leq n, j\neq 0}\log p(w_{t+j}|w_t) \tag{1.1}$$

unde:

- L este funcția obiectiv pe care dorim să o maximizăm. Este probabilitatea logaritmică medie a cuvintelor corecte din context, având în vedere cuvântul curent.
- T este numărul total de poziții din corpus.
- t se referă la poziția curentă pe care o luăm în considerare în corpus.
- w_t reprezintă cuvântul de la poziția t.
- w_{t+j} reprezintă cuvintele de context din jurul lui w_t.
- n reprezintă dimensiunea ferestrei de cuvinte contextuale din jurul cuvântului curent.
- $p(w_{t+j}|w_t)$ reprezintă probabilitatea condiționată a fiecărui cuvânt de context, având în vedere cuvântul curent w_t.

$p(w_{t+j}|w_t)$ se calculează folosind funcția softmax [5]:

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{j=1}^{V}\exp({v'_j}^T v_{w_I})} \tag{1.2}$$

unde:

- $p(w_O|w_I)$ este probabilitatea fiecărui cuvânt de la ieșire w_O, având în vedere cuvântul de la intrare w_I.
- v'_{w_O} este reprezentarea vectorială "de ieșire" a fiecărui cuvânt de ieșire w_O.
- v_{w_I} este reprezentarea vectorială "de intrare" a cuvântului de intrare w_I. Suma de la numitor se referă la toate cuvintele din vocabular (V).

Continuous Bag of Words (CBOW)

Modelul CBOW, ca și modelul Skip-gram, este o rețea neuronală superficială, însă este construit diferit. Acesta primește un set de cuvinte din context, fiecare dintre acestea fiind reprezentat ca un vector one-hot. Pentru a produce reprezentări de cuvinte, acestea sunt trecute printr-un strat de vectorizare. Reprezentările sunt apoi mediate sau însumate pentru a produce un singur vector ce reprezintă întregul context. Apoi vectorul este procesat de un strat de ieșire softmax, care generează o distribuție de probabilitate pentru fiecare cuvânt din vocabular. Cuvântul cu cea mai mare probabilitate este ales ca predicție a cuvântului țintă.

Funcția care trebuie maximizată pentru modelul CBOW este [5]:

$$L = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-n}, \ldots, w_{t+n}) \tag{1.3}$$

unde:

- L este funcția obiectiv pe care dorim să o maximizăm. Este probabilitatea logaritmică medie a cuvântului corect, având în vedere cuvintele din context.
- T este numărul total de poziții din corpul de instruire.
- t se referă la poziția curentă pe care o luăm în considerare în corpus.
- w_t reprezintă cuvântul de la poziția t.
- $w_{t-n}, ..., w_{t+n}$ reprezintă cuvintele din context din jurul lui w_t.
- $p(w_t|w_{t-n}, ..., w_{t+n})$ este probabilitatea condiționată a cuvântului w_t având în vedere cuvintele din context.

Probabilitatea condiționată în CBOW se calculează astfel [5]:

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{j=1}^{V} \exp({v'_j}^T v_{w_I})} \tag{1.4}$$

unde:

- $p(w_O|w_I)$ este probabilitatea cuvântului de ieșire w_O, având în vedere cuvintele de intrare w_I.
- v'_{w_O} este reprezentarea vectorială "de ieșire" a cuvântului de ieșire w_O.
- v_{w_I} este reprezentarea vectorială "de intrare" a cuvintelor de intrare w_I.

Deoarece funcția softmax din modelele skip-gram și CBOW este costisitoare din punct de vedere computațional, se utilizează abordări precum softmax ierarhic sau eșantionarea negativă.

Softmax Ierarhic

În stratul softmax standard, pentru fiecare eșantion de instruire, modelul calculează o probabilitate pentru fiecare cuvânt din vocabular - ceea ce poate fi costisitor din punct de vedere computațional pentru vocabularele mari.

Softmax ierarhic rezolvă acest aspect prin înlocuirea stratului softmax tradițional cu un arbore binar. Fiecare nod al arborelui reprezintă un cuvânt din vocabular. Pentru a calcula probabilitatea unui cuvânt, trebuie parcursă calea de la rădăcina arborelui până la cuvântul respectiv, calculând probabilitățile pe parcurs. Acest lucru reduce semnificativ costul de calcul, deoarece numărul de calcule este proporțional cu logaritmul dimensiunii vocabularului, nu cu dimensiunea vocabularului în sine [7].

Eșantionarea Negativă

O altă modalitate de a simplifica dificultatea computațională de a prezice un cuvânt țintă dintr-un vocabular mare este eșantionarea negativă.

Eșantionarea negativă schimbă obiectivul de la prezicerea cuvântului țintă în funcție de context (pentru CBOW) sau de prezicerea contextului în funcție de cuvânt (pentru Skip-gram) la o sarcină de clasificare binară. Modelul învață să diferențieze cuvântul țintă de câteva cuvinte "negative" alese aleatoriu pentru fiecare eșantion de antrenament [7].

Termenii "negativi" sunt aleși cu ajutorul unei distribuții de probabilitate, cuvintele mai frecvente având o probabilitate mai mare de a fi selectate ca cuvinte negative.

Eșantionarea negativă crește eficiența instruirii prin minimizarea numărului de ponderi care trebuie actualizate pentru fiecare eșantion de instruire [5].

1.3.2 FastText

FastText este o extensie a modelului Word2Vec pentru învățarea rapidă a reprezentărilor de cuvinte și clasificarea frazelor. A fost propus în 2016 de către laboratorul de cercetare AI al Facebook. În loc să învețe reprezentări pentru cuvinte individuale, FastText învață reprezentări pentru n-grame de caractere și reprezintă cuvintele ca sumă a vectorilor de n-grame [8].

FastText înlocuiește vectorii de cuvânt individuali v_{w_t} din funcția obiectivă Word2Vec cu suma reprezentărilor vectoriale ale n-gramelor de caractere din cuvânt. Dacă setul de n-grame de caractere dintr-un cuvânt w este notat cu G_w, iar reprezentarea vectorială a unei n-grame de caractere g cu z_g, atunci vectorul de cuvânt în FastText este dat de [8]:

$$v_w = \sum_{g \in G_w} z_g \tag{1.5}$$

unde:

- G_w este ansamblul tuturor n-gramelor de caractere ale cuvântului w.
- z_g este reprezentarea vectorială a unei n-grame de caractere g.

Prin urmare, modelul FastText înlocuiește obiectivul Word2Vec cu următorul [8]:

$$L = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|\sum_{g\in G_{w_t}} z_g) \tag{1.6}$$

unde:

- T este lungimea propoziției.
- t este poziția unui cuvânt în propoziție.
- c este dimensiunea ferestrei de context.
- w_{t+j} este un cuvânt din contextul cuvântului w_t.
- G_{w_t} este ansamblul tuturor n-gramelor de caractere ale cuvântului w_t.
- z_g este reprezentarea vectorială a unui n-gram de caractere g.

Probabilitatea condiționată se calculează astfel [8]:

$$p(w_{t+j}|\sum_{g\in G_{w_t}} z_g) = \frac{\exp((v'_{w_{t+j}})^T(\sum_{g\in G_{w_t}} z_g))}{\sum_{i=1}^{W}\exp((v'_i)^T(\sum_{g\in G_{w_t}} z_g))} \tag{1.7}$$

unde:

- $v'_{w_{t+j}}$ este reprezentarea vectorială de "ieșire" a cuvântului contextual.
- W este numărul total de cuvinte unice din corpus.
- G_{w_t} este ansamblul tuturor n-gramelor de caractere ale cuvântului w_t.
- z_g este reprezentarea vectorială a unei n-grame de caractere g.
- Simbolul $\sum_{g\in G_{w_t}} z_g$ reprezintă suma vectorilor n-gramelor din cuvântul w_t.

1.3.3 GloVe

GloVe, care provine de la "Global Vectors for Word Representation" (vectori globali pentru reprezentarea cuvintelor), este un algoritm de învățare nesupravegheată pentru a obține reprezentări vectoriale ale cuvintelor [9]. Acesta a fost dezvoltat de cercetători de la Stanford.

Obiectivul principal al modelului GloVe este de a minimiza diferența dintre produsul reprezentarilor vectoriale a două cuvinte și logaritmul numărului lor de co-ocurențe [10]:

$$V_i.V_j + b_i + b_j = log(X_{ij}) \tag{1.8}$$

unde

- V_i și V_j sunt reprezentările vectoriale ale cuvântului i și, respectiv, ale cuvântului j.
- b_i și b_j sunt termenii de bias pentru cuvintele i și, respectiv, j.
- X_{ij} este numărul de apariții ale cuvântului j în contextul cuvântului i.

Funcția pe care modelul GloVe urmărește să o minimizeze este [10]:

$$L = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij}) \left(V_i.V_j + b_i + b_j - log(X_{ij})\right)^2 \tag{1.9}$$

unde:

- L este funcția obiectiv de minimizat.
- V este numărul total de cuvinte din vocabular.
- $f(X_{ij})$ este o funcție de ponderare care atribuie o pondere mai mică co-ocurențelor mai puțin frecvente.
- V_i și V_j sunt reprezentările vectoriale ale cuvântului i și, respectiv, ale cuvântului j.
- b_i și b_j sunt termenii de bias pentru cuvintele i și, respectiv, j.
- X_{ij} este numărul de apariții ale cuvântului j în contextul cuvântului i.

În modelul GloVe, funcția de ponderare $f(X_{ij})$ este definită de obicei astfel:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{X_{max}}\right)^{3/4} & \text{if } X_{ij} < X_{max} \\ 1 & \text{otherwise} \end{cases} \tag{1.10}$$

unde:

- $f(X_{ij})$ este o funcție de ponderare care atribuie o pondere mai mică co-ocurențelor mai puțin frecvente.
- X_{ij} este numărul de apariții ale cuvântului j în contextul cuvântului i.
- X_{max} este un parametru de întrerupere, ales de obicei ca fiind 100.

1.3.4 ELMo

ELMo (Embeddings from Language Models) reprezintă o metodă de reprezentare vectorială a cuvintelor ce presupune antrenarea unui model lingvistic pe un corpus mare de text. Spre deosebire de metodele clasice, cum ar fi Word2Vec și GloVe, care generează un singur vector static pentru fiecare cuvânt, ELMo generează vectori în mod dinamic, pe baza contextului în care apar cuvintele, fiind astfel cunoscute ca reprezentări de cuvinte dependente de context.

ELMo se bazează pe un model cunoscut sub numele de LSTM bidirecțional (BiLSTM), care este o variantă de rețea neuronală recurentă (RNN) [11]. Partea "bidirecțională" din BiLSTM se referă la faptul că acesta este alcătuit din două LSTM: unul care procesează datele de la stânga la dreapta (forward LSTM) și unul care procesează datele de la dreapta la stânga (backward LSTM). Aceste LSTM sunt antrenate pentru a prezice următorul cuvânt dintr-o frază având în vedere cuvintele anterioare (pentru forward LSTM) sau cuvântul anterior având în vedere cuvinte următoare (pentru backward LSTM) [12].

O propoziție poate fi reprezentată ca o secvență de T elemente:

$$W = \{w_1, w_2, \ldots, w_T\} \tag{1.11}$$

Pentru fiecare element w_k [11]:

1. Elementul este reprezentat într-un vector x_k^{LM} folosind o rețea neurală convoluțională bazată pe caractere.
2. $x_1^{LM}, x_2^{LM}, \ldots, x_T^{LM}$ se aplică unui forward LSTM pentru a obține o secvență de stări ascunse $h_k^{fwd} = LSTM_{fwd}(h_{k-1}^{fwd}, x_k^{LM})$.
3. $x_T^{LM}, x_{T-1}^{LM}, \ldots, x_1^{LM}$ se aplică unui backward LSTM pentru a obține o secvență de stări ascunse $h_k^{bwd} = LSTM_{bwd}(h_{k+1}^{bwd}, x_k^{LM})$.
4. Concatenăm stările ascunse rezultate pentru a obține reprezentarea finală: $h_k = [h_k^{fwd}; h_k^{bwd}]$

La final, pentru fiecare element, avem 3 reprezentări: cea inițială x_k^{LM} și cele două reprezentări BiLSTM h_k^{fwd} și h_k^{bwd}.

Reprezentarea ELMo pentru un element este o sumă ponderată a acestor trei reprezentări, așa cum se poate observa în 1.2. Ponderile sunt învățate ca parte a procesului de antrenare și pot fi diferite pentru fiecare token, permițând modelului să accentueze aspecte diferite ale contextului pentru cuvinte diferite. Vectorul ELMo final pentru un anumit element w_k este calculat astfel [11]:

$$ELMo_k = \gamma \cdot (s_0 \cdot x_k^{LM} + s_1 \cdot h_k^{fwd} + s_2 \cdot h_k^{bwd}) \tag{1.12}$$

unde:

- γ este un scalar care controlează scala vectorului ELMo
- s_i sunt greutăți normalizate softmax

1.3.5 BERT

BERT, sau Bidirectional Encoder Representations from Transformers, este o tehnică dezvoltată și lansată de Google pentru pre-antrenarea modelelor de procesare a limbajului natural (NLP). Datorită performanțelor sale remarcabile, BERT a avut o influență majoră asupra sarcinilor NLP. BERT folosește transformer-ul, un mecanism de atenție care învață relațiile contextuale între cuvintele dintr-un text [14].

Spre deosebire de alte metode de reprezentare (cum ar fi Word2Vec), BERT analizează contextul din ambele părți (stânga și dreapta cuvântului) și utilizează această înțelegere mai amplă pentru a genera reprezentarea vectorială a unui cuvânt [15]. Acesta este motivul pentru care este denumit "bidirecțional". Algoritmul BERT constă în următoarele etape fundamentale [14]:

- **Reprezentările de intrare:** BERT primește ca intrare un șir de cuvinte. Similar cu Word2Vec, fiecare cuvânt este convertit într-un vector numeric de dimensiune fixă.

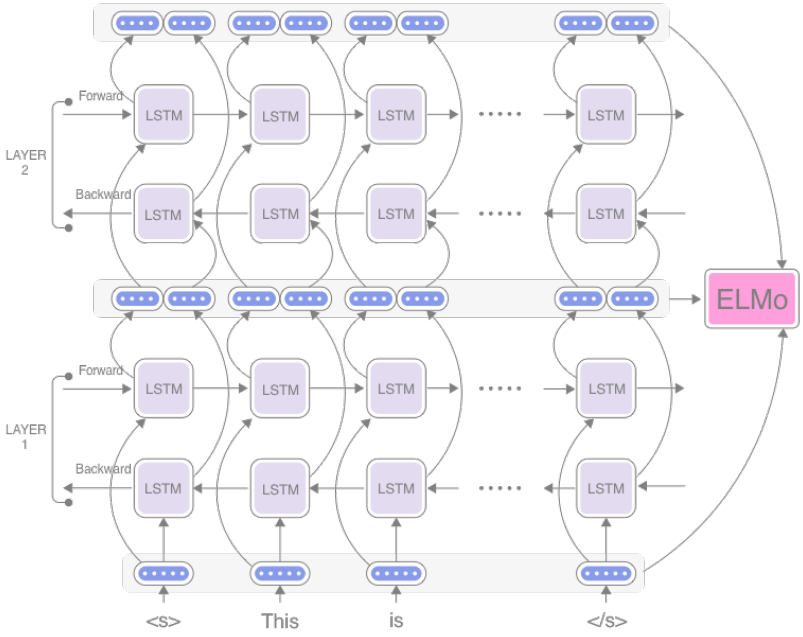


Figura 1.2: Arhitectura ELMo [13]

- **Codificarea poziţională:** BERT trebuie să înţeleagă şi ordinea cuvintelor dintr-o frază. Acest lucru se realizează prin includerea unei "codificări poziţionale" în reprezentările cuvintelor. Acest lucru presupune că încorporarea fiecărui cuvânt este modificată pentru a corespunde poziţiei acestuia, oferind modelului BERT informaţii despre ordinea cuvintelor.

- **Mecanismul de self-attention**: BERT utilizează un transformer, care este un mecanism de atenţie ce învaţă relaţiile contextuale dintre cuvintele dintr-un text. Cu alte cuvinte, în loc să observe cuvintele în mod izolat, acesta analizează modul în care acestea se raportează la alte cuvinte din frază. Acest lucru este denumit "atenţie" [16].

- **Bidirecţionalitate:** Metodele tradiţionale de înţelegere a textului analizează cuvintele într-o singură direcţie (de la stânga la dreapta sau de la dreapta la stânga). BERT, pe de altă parte, analizează datele de intrare în ambele direcţii, captând mai multe informaţii de context.

- **Instruire:** BERT este instruit folosind două metode nesupervizate: Masked Language Model (MLM) şi Next Sentence Prediction (NSP). În cadrul metodei MLM, cuvinte aleatorii din propoziţie sunt înlocuite cu un simbol de mascare, iar modelul trebuie să prezică cuvântul original pe baza contextului. Pentru sarcina NSP, BERT învaţă să prezică dacă o propoziţie urmează unei alte propoziţii într-un text, ceea ce îl ajută să înţeleagă relaţiile dintre propoziţii.

1.4 Algoritmi

1.4.1 Naive Bayes

Naive Bayes este o familie de clasificatori probabilistici care se bazează pe aplicarea teoremei lui Bayes la caracteristici cu ipoteze de independență ridicată (naivă). Se numește naïve deoarece presupune că prezența unei caracteristici într-o clasă nu are nimic de-a face cu prezența altei caracteristici [17]. În practică, aceste ipoteze pot fi incorecte, deși algoritmul tinde să aibă performanțe bune chiar și atunci când aceste ipoteze sunt încălcate într-o anumită măsură.

Teorema lui Bayes

Teorema lui Bayes se află în centrul clasificatorului Naive Bayes. În versiunea sa cea mai elementară, teorema lui Bayes oferă o metodă de calculare a probabilității ca un punct de date să aparțină unei clase, având în vedere cunoștințele anterioare. Teorema lui Bayes este exprimată matematic astfel [18]:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{1.13}$$

unde:

- $P(A|B)$ este probabilitatea ipotezei A, având în vederecă ipoteza B era adevărată.
- $P(B|A)$ este probabilitatea datelor B, având în vedere că ipoteza A era adevărată.
- $P(A)$ este probabilitatea ca ipoteza A să fie adevărată (indiferent de ipoteză). Aceasta se numește probabilitatea anterioară a lui A.
- $P(B)$ este probabilitatea datelor (indiferent de ipoteză).

Clasificatorul Naive Bayes

Putem aplica teorema lui Bayes la o problemă în contextul învățării automate și al clasificării. Se dorește să se calculeze probabilitatea ca X să aparțină unei anumite clase c_i, având în vedere un vector de caracteristici $X = (x_1, x_2, \ldots, x_n)$ (reprezentând n caracteristici) și o variabilă de clasă C (cu clasele potențiale $c_1, c_2, \ldots, c_m$). Folosind teorema lui Bayes, aceasta se exprimă astfel [17]:

$$P(C = c_i|X) = \frac{P(X|C = c_i) * P(C = c_i)}{P(X)} \tag{1.14}$$

unde:

- $P(C = c_i|X)$ este probabilitatea posterioară a clasei c_i, având în vedere un predictor (caracteristici).
- $P(X|C = c_i)$ este probabilitatea care reprezintă probabilitatea predictorului dată de clasă.
- $P(C = c_i)$ este probabilitatea anterioară a clasei.
- $P(X)$ este probabilitatea anterioară a predictorului.

Se calculează probabilitatea posterioară $P(C = c_i|X)$ pentru fiecare clasă c_i în și se selectează clasa cu cea mai mare probabilitate posterioară.

$P(C = c_i)$ este frecvența relativă a clasei c_i în setul de instruire.

Probabilitatea $P(X|C = c_i)$ este mai dificil de calculat, deoarece este probabilitatea combinată a tuturor caracteristicilor în funcție de clasă. Atunci când există multe caracteristici, probabilitatea ca o anumită combinație să apară în setul de instruire devine foarte scăzută, ceea ce duce la probabilități zero care pot afecta modelul. Aici intervine ipoteza "naivă": se presupune că, dată fiind clasa, toate caracteristicile sunt independente condiționat [17]. Deși nu este întotdeauna cazul, aceasta simplifică substanțial calculul și funcționează foarte bine pentru o gamă largă de informații din lumea reală.

Având în vedere ipoteza naivă, probabilitatea se simplifică de la probabilitatea comună a tuturor caracteristicilor la produsul probabilităților individuale [17]:

$$P(X|C = c_i) = P(x_1|C = c_i) * P(x_2|C = c_i) * \ldots * P(x_n|C = c_i) \tag{1.15}$$

Tipuri de Clasificatoare Naive Bayes

Modul în care se calculează $P(x_i|C = c_i)$ depinde de natura datelor [19]:

- **Gaussian Naive Bayes:** Atunci când predictorii sunt mai degrabă continui decât discreți, se presupune că aceștia sunt extrași dintr-o distribuție gaussiană. Funcția de densitate gaussiană este apoi utilizată pentru a calcula probabilitatea $P(x_i|C = c_i)$.
- **Multinomial Naive Bayes:** Această metodă este utilizată atunci când datele sunt distribuite multinomial, cum ar fi în problemele de clasificare a textelor, unde predictorii sunt frecvența termenilor din document.
- **Bernoulli Naive Bayes**: Se utilizează atunci când predictorii sunt variabile binare.

1.4.2 Regresia Logistică

Regresia logistică este un model statistic utilizat cu precădere pentru probleme de clasificare binară. Această metodă presupune că există o relație probabilistică între variabilele caracteristice (variabile independente) și variabila țintă (variabila dependentă) care poate fi descrisă cu precizie cu ajutorul unei funcții logistice sau a unei funcții sigmoide. Funcția sigmoidă este capabilă să transforme orice număr cu valoare reală într-o valoare cuprinsă între 0 și 1 [20]:

$$p(x) = \frac{1}{1 + e^{-z}} \tag{1.16}$$

unde:

- $p(x)$ este probabilitatea clasei implicite (clasa țintă pe care dorim să o prezicem).
- z este suma ponderată a intrărilor în funcție, care cuprinde variabilele independente sau caracteristicile dumneavoastră. Aceasta este dată de: $z = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$

$\beta_0, \beta_1, \ldots, \beta_n$ sunt parametrii (coeficienții) modelului. Aceste valori necesită o estimare pentru a permite algoritmului să modeleze cel mai bine date.

Rezultatul modelului este probabilitatea ca o anumită intrare să aparțină unei anumite clase. Dacă probabilitatea depășește un anumit prag (de obicei 0.5), punctul este atribuit clasei implicite. În caz contrar, este considerat ca aparținând clasei opuse.

Deoarece răspunsul în regresia logistică nu este continuu, parametrii nu sunt estimați cu ajutorul regresiei standard a celor mai mici pătrate. În schimb, se utilizează estimarea probabilității maxime [17]:

$$\log(L) = \sum_{i=1}^{n} [y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))] \tag{1.17}$$

unde:

- y_i este eticheta reală a clasei (0 sau 1).
- $p(y_i)$ este probabilitatea prezisă.

Scopul este de a găsi $\beta_0, \beta_1, \ldots, \beta_n$ care maximizează probabilitatea logaritmică. După identificarea coeficienților modelului, aceștia pot fi utilizați pentru a estima probabilitatea ca noile date să aparțină unei clasei.

1.4.3 Support Vector Machine

SVM-urile sunt modele de învățare supervizată care sunt utilizate pentru analiza de clasificare și regresie. Acestea fac parte din familia clasificatoarelor liniare generalizate și pot fi considerate ca o extensie a perceptronului. Acestea funcționează prin generarea unui hiperplan sau a unui grup de hyperplanuri într-un spațiu înalt sau infinit-dimensional care poate fi utilizat pentru clasificare sau regresie [21].

Înainte de a explica algoritmul SVM, trebuie definite anumite noțiuni fundamentale [22]:

- **Hiperplan:** Un hiperplan este un subspațiu a cărui dimensiune este cu unu mai mică decât cea a spațiului său ambiant. Într-un spațiu bidimensional, un hiperplan este o linie; într-un spațiu tridimensional, este un plan, și așa mai departe.
- **Vectori de suport:** Vectorii de suport reprezintă acele puncte de date din setul de antrenare care se află cel mai aproape de granița de decizie a SVM. Aceste puncte sunt esențiale pentru determinarea poziției granței și a marginii, deoarece ele influențează în mod direct construcția și performanța clasificatorului.
- **Margine:** În contextul SVM, termenul margine se referă la distanța dintre linia de separare și cele mai apropiate puncte de date din fiecare clasă. Aceasta reprezintă separarea pe care SVM încearcă să o maximizeze pentru a îmbunătăți generalizarea și performanța clasificatorului pe datele necunoscute.

În cazul claselor liniar separabile, algoritmul SVM urmărește să găsească hiperplanul optim care separă clasele. Din punct de vedere matematic, un hiperplan este descris ca [22]:

$$\mathbf{w} \cdot \mathbf{x} - b = 0 \tag{1.18}$$

unde:

– w este vectorul de ponderare

– x este un eșantion de intrare

– b este distorsiunea.

Funcția de decizie pentru o nouă instanță x' devine [22]:

$$f(\mathbf{x}') = \mathbf{w} \cdot \mathbf{x}' - b \tag{1.19}$$

Dacă rezultatul este pozitiv, exemplul aparține clasei pozitive, iar dacă este negativ, acesta aparține clasei negative.

Scopul este de a alege w și b care să maximizeze marginea dintre clase. Marginea este de două ori distanța de la hiperplan la cele mai apropiate instanțe. Astfel maximizarea lui M este echivalentă cu minimizarea lui $||w||$ [22]:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \tag{1.20}$$

Regularizarea este o tehnică de evitare a overfitting-ului. Aceasta este controlată de un parametru notat cu C în SVM. O valoare C mai mică determină o margine mai largă, dar mai multe clasificări eronate (crește bias-ul, dar scade varianța). O valoare mai mare a C generează o margine mai mică și permite mai puține clasificări eronate (bias mai mic, dar variație mai mare) [21]:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i \tag{1.21}$$

Multe clase nu sunt liniar separabile. În acest caz, se proiectează datele într-un spațiu cu dimensiuni mai mari în care acestea pot fi separate liniar. Pentru această proiecție se utilizează o funcție kernel $K(x, x')$. Cele mai comune funcții kernel sunt [22]:

1. **Liniar:** $K(x, x') = x \cdot x'$
2. **Polinomial:** $K(x, x') = (\gamma \cdot x \cdot x' + r)^d$
3. **Radial Basis:** $K(x, x') = \exp(-\gamma \cdot ||x - x'||^2)$
4. **Sigmoid:** $K(x, x') = \tanh(\gamma \cdot x \cdot x' + r)$

În aceste formule, γ controlează influența produsului asupra calculului nucleului, d este gradul polinomului și r este o constantă opțională, utilă pentru deplasarea limitei de decizie.

Capitolul 2

Descrierea Implementării

2.1 Setul de Date

Pentru realizarea acestui proiect s-a ales folosirea bazei de date Hateval 2019, care constă în detectarea discursului instigator la ură pe Twitter, prezentat pe două ținte specifice diferite, imigranții și femeile, într-o perspectivă multilingvă, în spaniolă și engleză.

Tweeturile au fost adunate folosind o combinație de verificare manuală și selecție algoritmică. Metoda de verificare umană a inclus căutarea de tweet-uri de la conturi cunoscute de discursuri de ură și utilizarea de cuvinte-cheie și hashtag-uri legate de discursuri de ură. Procedura de filtrare automată include identificarea tweet-urilor care puteau conține discursuri instigatoare la ură, utilizând un model de identificare a discursurilor instigatoare la ură preformat [23].

Datele au fost furnizate în modul descris în figurile 2.1 și 2.2, în funcție de limbile (spaniolă și engleză) și de grupurile țintă (femei și imigranți). Setul de date în spaniolă cuprinde 5.000 de tweet-uri (3.209 pentru femeile țintă și 1.991 pentru imigranți), în timp ce setul în engleză cuprinde 10.000 de tweet-uri (5.000 pentru fiecare țintă) [23].

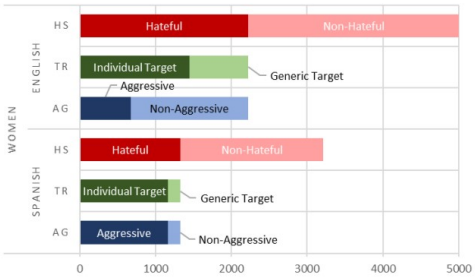


Figura 2.1: Femei - Hateval-2019 [23]

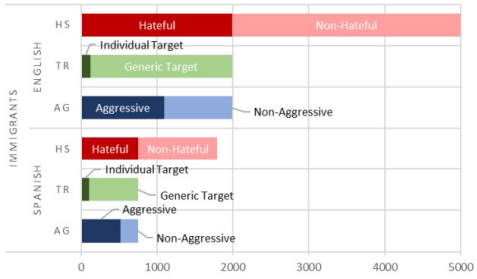


Figura 2.2: Imigranți - Hateval-2019 [23]

Primele 6 rânduri din setul de date Hateval-2019 pot fi observate in figura 2.3. Acesta are următoarele coloane [23]:

- ID - numărul de identificare al fiecărei instanțe din setul de date
- text - Tweet-ul în limba engleză sau spaniolă, în funcție de setul de date ales

- HS (Hate Speech) - o valoare binară care indică dacă discursul instigator la ură este îndreptat împotriva uneia dintre țintele date (femei sau imigranți): 1 dacă se întâmplă, 0 dacă nu.
- TR (Target Range) - în cazul în care apare HS (adică valoarea caracteristicii HS este 1), o valoare binară care indică dacă ținta este un grup generic de persoane (0) sau o anumită persoană (1).
- AG (Aggressiveness) - în cazul în care apare HS (adică valoarea pentru caracteristica HS este 1), o valoare binară care indică dacă tweeterul este agresiv (1) sau nu (0)..

d	text	HS	TR	AG
201	Hurray, saving us $$$ in so many ways @potus @realDonaldTrump #LockThemUp #BuildTheWall #EndDACA #BoycottNFL #BoycottNike	1	0	0
202	Why would young fighting age men be the vast majority of the ones escaping a war & not those who cannot fight like women, children, and the elderly?It's because the majority of the refugees are not actually refugees they are economic migrants trying to get into Europe.... https://t.co/Ks0SHbtYqn	1	0	0
203	@KamalaHarris Illegals Dump their Kids at the border like Road Kill and Refuse to Unite! They Hope they get Amnesty, Free Education and Welfare Illegal #FamilesBelongTogether in their Country not on the Taxpayer Dime Its a SCAM #NoDACA #NoAmnesty #SendThe	1	0	0
204	NY Times: 'Nearly All White' States Pose 'an Array of Problems' for Immigrants https://t.co/ACZKLhdMV9 https://t.co/CJAlSXCzR6	0	0	0
205	Orban in Brussels: European leaders are ignoring the will of the people, they do not want migrants https://t.co/NeYFyqvYlX	0	0	0
206	@KurtSchlichter LEGAL is. Not illegal. #BuildThatWall	1	0	0

Figura 2.3: Setul de date Hateval-2019

2.2 Detalii despre Implementare

– Mediul utilizat pentru implementarea acestui proiect a fost Google Colab.

– Pentru versiunea GPU, a fost alocat un Tesla T4, mai multe detalii despre acesta pot fi văzute în figura 3.5.

```
Sun May 21 17:48:18 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   66C    P8    10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figura 2.4: Detalii GPU

– Pentru cei 4 algoritmi implementați, dimensiunile de intrare sunt diferite în funcție de metoda de reprezentare. Acest lucru este util pentru a observa modul în care dimensiunea de intrare afectează timpul de execuție și utilizarea GPU.

Algoritm	ELMo	FastText	Word2Vec	GloVe	Bert
Dimensiune	768	300	300	50	768

Tabela 2.1: Dimensiunea vectorilor în funcție de metoda de reprezentare

- Hiperparametrii folosiți:
  - **Gaussian Naive Bayes:** 'verbose': 4, 'output_type': 'input', 'priors': None, 'var_smoothing': 1e-09
  - **Bernoulli Naive Bayes:** 'verbose': 4, 'output_type': 'input', 'alpha': 1.0, 'fit_prior': True, 'class_prior': None, 'binarize': 0.0
  - **Logistic Regression:** 'verbose': 4, 'output_type': 'input', 'penalty': 'l2', 'tol': 0.0001, 'C': 1.0, 'fit_intercept': True, 'class_weight': None, 'max_iter': 1000, 'linesearch_max_iter': 50, 'l1_ratio': None, 'solver': 'qn'
  - **SVM:** 'verbose': 4, 'output_type': 'input', 'C': 1, 'kernel': 'rbf', 'degree': 3, 'gamma': 'scale', 'coef0': 0.0, 'tol': 0.001, 'cache_size': 1024.0, 'max_iter': -1, 'nochange_steps': 1000, 'probability': False, 'random_state': None, 'class_weight': None, 'multiclass_strategy': 'ovo'
  - **Random Forest:** 'verbose': 4, 'output_type': 'input', 'n_estimators': 100, 'max_depth': 5, 'max_features': 'auto', 'n_bins': 128, 'split_criterion': 0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_impurity_decrease': 0.0, 'bootstrap': True, 'max_samples': 1.0, 'max_leaves': -1, 'accuracy_metric': None, 'max_batch_size': 4096, 'n_streams': 4, 'dtype': dtype('float64'), 'min_weight_fraction_leaf': None, 'n_jobs': None, 'max_leaf_nodes': None, 'min_impurity_split': None, 'oob_score': None, 'random_state': None, 'warm_start': None, 'class_weight': None, 'criterion': None

2.3 Etapele Implementării

1. **Configurarea mediului:** În primul rând, codul se asigură că este disponibilă o placă grafică (GPU) și că bibliotecile necesare RAPIDS sunt instalate. Verifică tipul GPU-ului și îi cere utilizatorului să schimbe tipul de runtime la GPU, dacă este necesar.
2. **Montarea unității Drive și importarea bibliotecilor:** Unitatea Google Drive este setată pentru a accesa fișierele necesare. Se importă bibliotecile necesare pentru cod, inclusiv transformers pentru procesarea limbajului natural și gpustat pentru monitorizarea GPU-ului.
3. **Selectarea setului de date și metoda de reprezentare vectorială:** Utilizatorului i se solicită să aleagă un set de date și o metodă de reprezentare. Opțiunile disponibile pentru setul de date sunt "Hateval2019" și "Davidson", în timp ce metoda de reprezentare poate fi selectată din "ELMo", "FastText", "Word2Vec", "GloVe" sau "BERT".
4. **Prelucrarea datelor**: Datele textuale din setul de date selectat sunt prelucrate prin eliminarea etichetelor HTML, URL-urilor, tokenizarea textului, convertirea to-

kenilor la litere mici, eliminarea cuvintelor de oprire (stopwords) și lematizarea tokenilor. Acești pași ajută la curățarea și normalizarea textului, pentru a face ușura procesul de a le analiza.

5. **Încărcarea metodelor de reprezentare vectorială:** Sunt încărcate modelele pre-antrenate pentru reprezentarea cuvintelor corespunzătoare metodei de vectorizare selectate. Metodelde vectorizare oferă o reprezentare numerică a cuvintelor din text, capturând informații semantice.
6. **Împărțirea datelor:** Setul de date este împărțit în seturi de antrenare și de testare, folosind o rată de 80:20.
7. **Algoritmi de învățare automată:** Codul implementează mai mulți algoritmi de învățare automată utilizând biblioteca cuML, care furnizează versiuni accelerate pe GPU ale algoritmilor populare de învățare automată. Algoritmii implementați includ Naive Bayes Gaussian, Naive Bayes Bernoulli, regresie logistică, SVM și Random Forests.
8. **Evaluare:** Se calculează și se afișează acuratețea fiecărui algoritm atât pentru setul de antrenare, cât și pentru cel de testare. Acuratețea măsoară cât de bine se descurcă modelele în clasificarea datelor textuale în categorii predefinite.

Pentru a evalua impactul utilizării GPU-ului, a fost dezvoltată și o implementare complementară bazată pe CPU în scopul analizei comparative. În această abordare alternativă, biblioteca cuML a fost înlocuită cu biblioteca sklearn, în timp ce biblioteca cuDF a fost înlocuită cu pandas. Au fost efectuate evaluări asupra ambelor implementări, incluzând măsurători ale timpului de execuție, acuratețe. În cazul versiunii cu GPU, s-a măsurat și utilizarea GPU-ului.

Capitolul 3

Rezultate

3.1 Rezultate

3.1.1 GPU

În următoarele tabele sunt prezentate rezultatele obținute prin antrenarea algoritmilor Gaussian Naive Bayes, Bernoulli Naive Bayes, Logistic Regression, Support Vector Machines și Random Forest pe GPU. Au fost măsurați timpii pe seturile de antrenare și de testare, acuratețea pe setul de antrenare și de testare și procentul de utilizare al unității de procesare grafică (GPU).

	Timp Train	Acuratețe Train	Utilizare GPU	Timp Test	Acuratețe Test
GNB	0.78	61.29	4%	4.07	59.92
BNB	1.95	62.18	3%	0.43	59.92
LR	2.89	78.07	71%	1.54	71.32
SVM	1.76	81.98	68%	0.37	73.35
RF	0.61	70.23	51%	0.6	65.12

Tabela 3.1: ELMo GPU

	Timp Train	Acuratețe Train	Utilizare GPU	Timp Test	Acuratețe Test
GNB	0.67	61.48	6%	1.89	59.62
BNB	0.26	61.18	2%	0.27	59.62
LR	0.39	68.12	10%	0.26	66.65
SVM	1.33	74.36	99%	0.37	70.85
RF	0.46	66.72	48%	0.36	64.23

Tabela 3.2: FastText GPU

	Timp Train	Acuratețe Train	Utilizare GPU	Timp Test	Acuratețe Test
GNB	0.84	59.13	3%	2.61	57.54
BNB	0.29	59.84	4%	0.3	57.54
LR	0.41	71.51	26%	0.32	69.46
SVM	1.41	79.17	99%	0.41	72.92
RF	0.47	67.18	67%	0.36	64.19

Tabela 3.3: Word2Vec GPU

	Timp Train	Acuratețe Train	Utilizare GPU	Timp Test	Acuratețe Test
GNB	0.95	57.39	2%	2.64	54.81
BNB	0.34	57.28	3%	0.25	54.81
LR	0.51	62.73	26%	0.26	61.73
SVM	0.96	65.48	68%	0.34	63.96
RF	0.58	65.45	44%	0.4	61.62

Tabela 3.4: GloVe GPU

	Timp Train	Acuratețe Train	Utilizare GPU	Timp Test	Acuratețe Test
GNB	0.75	64.47	5%	1.79	62.96
BNB	0.33	64.63	6%	0.34	62.96
LR	0.79	77.78	72%	0.32	72.46
SVM	0.49	76.18	81%	0.33	73.58
RF	0.54	73.06	66%	0.51	68.12

Tabela 3.5: BERT GPU

3.1.2 CPU

În scopul efectuării unei comparații între utilizarea unității de procesare grafică (GPU) și a unității centrale de procesare (CPU) în contextul aplicațiilor de învățare automată (machine learning), se prezintă în continuare rezultatele obținute prin antrenarea algoritmilor Gaussian Naive Bayes, Bernoulli Naive Bayes, Logistic Regression, Support Vector Machines și Random Forest pe CPU. Informațiile sunt prezentate succint în tabelele de mai jos, includând timpii de antrenare și de testare, precum și acuratețea obținută pe seturile de antrenare și de testare.

	Timp Train	Acuratețe Train	Timp Test	Acuratețe Test
GNB	0.07	61.29	0.03	59.92
BNB	0.21	62.18	0.06	60.73
LR	1.37	77.95	0.01	70.96
SVM	56.31	81.98	17.97	73.35
RF	13.41	70.85	0.05	65.5

Tabela 3.6: ELMo CPU

	Timp Train	Acurateţe Train	Timp Test	Acurateţe Test
GNB	0.04	61.48	0.01	59.62
BNB	0.06	61.18	0.03	60.62
LR	0.33	68.12	0.01	66.73
SVM	14.65	74.35	5.58	70.85
RF	7.1	67.51	0.05	64.27

Tabela 3.7: FastText CPU

	Timp Train	Acurateţe Train	Timp Test	Acurateţe Test
GNB	0.07	59.13	0.02	57.54
BNB	0.07	59.84	0.02	57.73
LR	0.32	71.55	0.01	69.42
SVM	28.62	79.15	7.14	72.92
RF	8.1	67.99	0.05	64.73

Tabela 3.8: Word2Vec CPU

	Timp Train	Acurateţe Train	Timp Test	Acurateţe Test
GNB	0.01	57.39	0.01	54.81
BNB	0.02	57.28	0.01	54.27
LR	0.3	62.74	0.01	61.58
SVM	7.31	65.47	2.72	63.96
RF	5.67	65.57	0.05	62.08

Tabela 3.9: GloVe CPU

	Timp Train	Acurateţe Train	Timp Test	Acurateţe Test
GNB	0.06	64.47	0.02	62.96
BNB	0.17	64.62	0.05	63.5
LR	1.21	77.41	0.01	73.23
SVM	61.34	76.17	15.68	73.58
RF	13.07	72.89	0.05	68.81

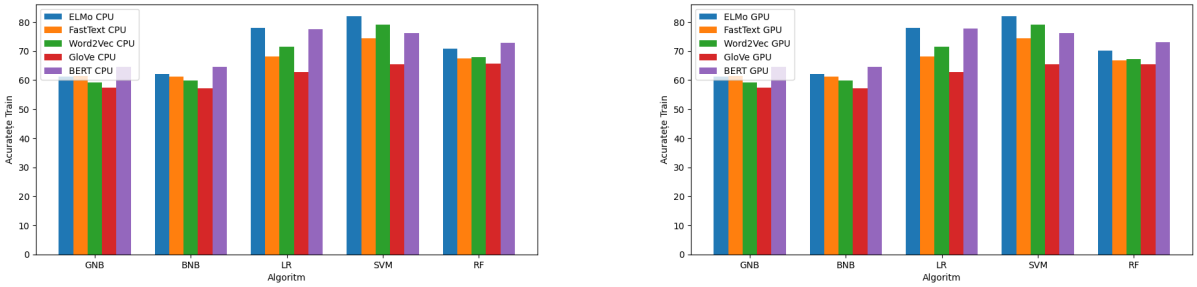
Tabela 3.10: BERT CPU

3.2 Comparaţie

3.2.1 Acurateţe

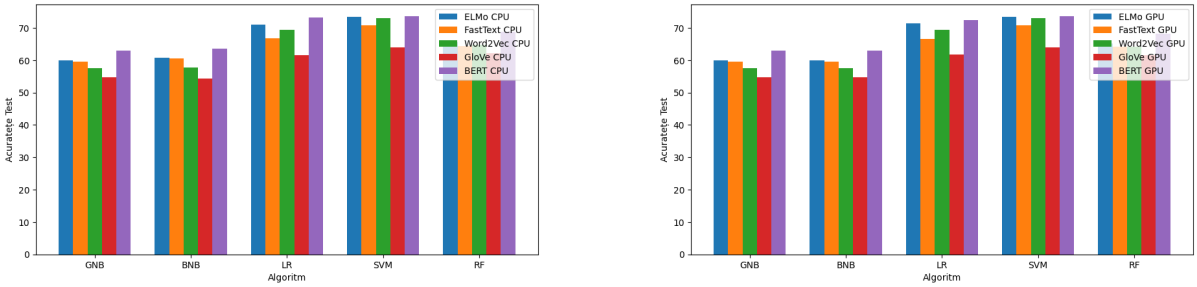
Algoritmul SVM are în general cea mai mare acurateţe la antrenare dintre metodele testate pe diferite reprezentări vectoriale. Acesta are cele mai bune rezultate cu reprezentările ELMo atât pe arhitecturile GPU, cât şi CPU, obţinând o acurateţe la antrenare de 81,98%.

Totuşi, este important de subliniat că o acurateţe ridicată la antrenare nu înseamnă neapărat o acurateţe ridicată la testare din cauza riscului de overfitting. Acurateţea la testare oferă o măsură mai bună a performanţei modelului pe date nevăzute. În acest sens, SVM cu reprezentările ELMo demonstrează, de asemenea, cea mai mare acurateţe la testare pe ambele arhitecturi, atingând o acurateţe de 73,35% pe GPU şi o acurateţe similară de 73,35% pe CPU.



(a) CPU (b) GPU

Figura 3.1: Acuratețea (Train) în funcție de algoritm si de metoda de reprezentare



(a) CPU (b) GPU

Figura 3.2: Acuratețea (Test) în funcție de algoritm si de metoda de reprezentare

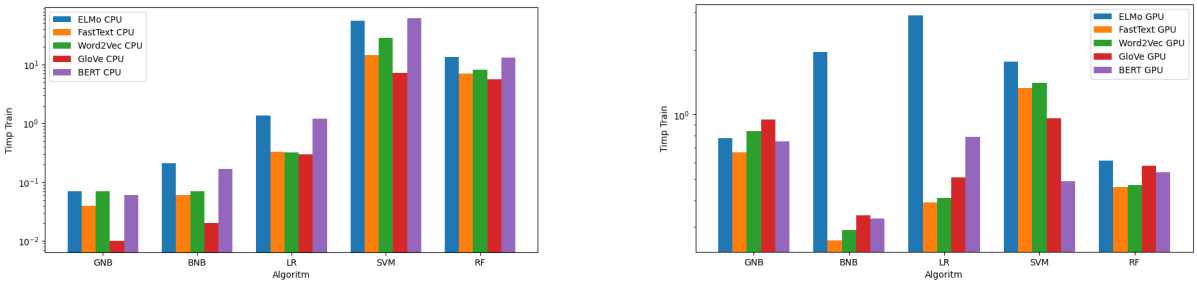
3.2.2 Metode de Reprezentare Vectorială

Privind rezultatele, ELMo și BERT în general performează mai bine decât FastText, Word2Vec și GloVe în ceea ce privește acuratețea. Cu toate acestea, acestea necesită mai mult timp și utilizează mai multe resurse GPU. În contrast, FastText, Word2Vec și GloVe sunt mai rapide și utilizează mai puține resurse GPU, dar oferă o acuratețe mai mică.

3.2.3 Timp de Execuție

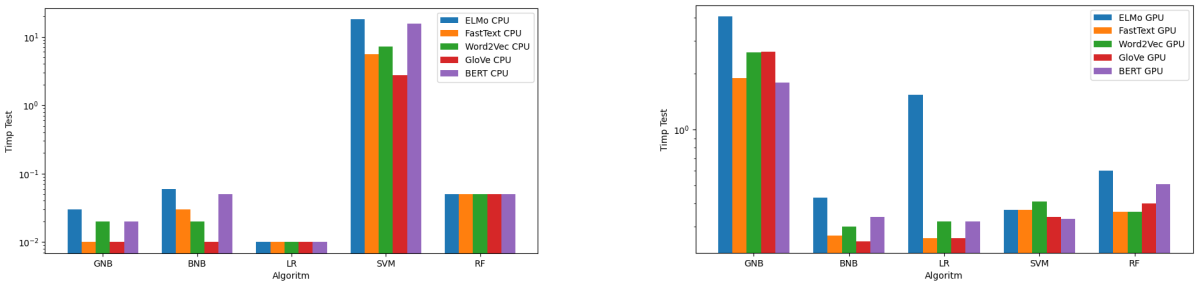
În general, timpul de antrenare și testare al modelului este mai mic atunci când se utilizează un GPU în comparație cu un CPU. Acest lucru este deosebit de pronunțat pentru algoritmii SVM și Random Forest (RF) cu majoritatea metodelor de vectorizare, care pot fi atribuite naturii paralelizabile a acestor modele care beneficiază semnificativ de arhitectura GPU.

CPU durează considerabil mai mult pentru a antrena SVM cu toate metodele de vectorizare, cu timpul maxim de antrenare fiind de 61,34 secunde cu încorporările BERT. În contrast, pe un GPU, SVM durează mai puțin timp, cu maximul fiind de 1,76 secunde cu încorporările ELMo.



(a) CPU (b) GPU

Figura 3.3: Timpul de execuție (Train) în funcție de algoritm si de metoda de reprezentare



(a) CPU (b) GPU

Figura 3.4: Timpul de execuție (Test) în funcție de algoritm si de metoda de reprezentare

3.2.4 Utilizarea GPU

După cum era de așteptat, utilizarea GPU este mai mare pentru modele mai complexe, cum ar fi SVM, random forest și regresia logistică, cu majoritatea metodelor de reprezentare vectorială. În cazul algoritmului random forest, GPU-ul este cel mai utilizat pentru Word2Vec (67%) și BERT (66%). La regresia logistica, GPU-ul este utilizat în proporție de 71% pentru ELMo și 72% pentru BERT. Pentru SVM, se atinge o utilizare maximă a GPU-ului de 99% pentru metodele de reprezentare FastText și Word2Vec.

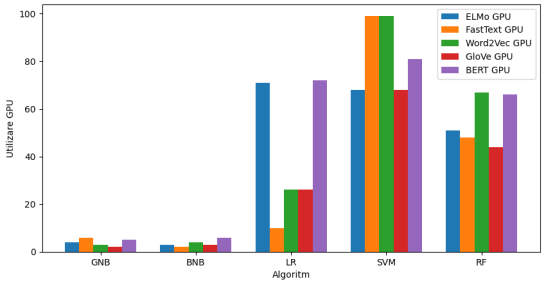


Figura 3.5: Procentul de utilizare GPU în funcție de algoritm si de metoda de reprezentare

Capitolul 4

Concluzii

- **Timp de execuție:** GPU-urile reduc semnificativ timpul de antrenament, în special pentru SVM, datorită capacităților sale de procesare paralelă. Cu toate acestea, pentru modele mai simple precum Naive Bayes, accelerarea GPU este mai puțin pronunțată.
- **Utilizarea GPU:** Modelele complexe precum SVM și regresia logistică utilizează GPU-ul mai intensiv. Metodele de vectorizare a textului influențează de asemenea utilizarea GPU, indicând complexitatea lor computațională.
- Acuratețe: Trecerea de la CPU la GPU nu modifică semnificativ acuratețea modelului. În schimb, alegerea modelului și metoda de vectorizare a textului joacă roluri mai importante în determinarea acurateței.
- **Metoda de Reprezentare Vectorială** ELMo și BERT oferă o acuratețe mai mare, dar necesită mai multă utilizare a GPU și timp de antrenament. FastText, Word2Vec și GloVe oferă un compromis cu o acuratețe mai scăzută, dar o cerere computațională mai mică.
- **Modelul de Machine Learning:** Regresia logistică, ranfom forest și SVM beneficiază cel mai mult de accelerarea GPU, arătând timpi de antrenament mai rapizi și o acuratețe similară cu versiunile lor CPU. Beneficiile accelerării GPU sunt mai puțin evidente în modelele Naive Bayes.
- **Optimizarea resurselor:** Alegerea metodei de reprezentare a textului și a modelului de machine learning trebuie să echilibreze obiectivele de acuratețe cu constrângerile de resurse, cum ar fi utilizarea GPU și timpul de execuție.
- **Utilizarea GPU:** Beneficiile accelerării GPU sunt mai pronunțate în sarcinile computațional intensive. Prin urmare, pentru modele mai simple sau seturi de date mai mici, accelerarea GPU s-ar putea să nu fie întotdeauna alegerea optimă.

Bibliografie

[1] Mitchell T.M. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[2] Alpaydin E. *Introduction to Machine Learning*. MIT press, 2020.

[3] What is Machine Learning? `https://developers.google.com/machine-learning/intro-to-ml/what-is-ml`. Accessed: 2022-06-11.

[4] Abdi H. and Williams L. J. Principal Component Analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[5] Rong X. word2vec Parameter Learning Explained. *arXiv preprint arXiv:1411.2738*, 2014.

[6] Mikolov T. et al. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[7] Goldberg Y. and Levy O. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. *arXiv preprint arXiv:1402.3722*, 2014.

[8] Bojanowski P. et al. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[9] GloVe: Global Vectors for Word Representation. `https://nlp.stanford.edu/projects/glove/`. Accessed: 2023-05-19.

[10] Pennington J. et al. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[11] Peters M. E. et al. Deep Contextualized Word Representations. *arXiv preprint arXiv:1802.05365*, 2018.

[12] Huang Z. et al. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[13] Confusion Matrix. `https://iq.opengenus.org/elmo/`. Accessed: 2023-05-17.

[14] Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Lee J. and Toutanova K. Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Vaswani A. et al. Attention is All You Need. *Advances in neural information processing systems*, 30, 2017.

[17] Murphy K. P. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[18] Gelman A. et al. *Bayesian Data Analysis*. CRC press, 2013.

[19] Raschka S. Naive Bayes and Text Classification Introduction and Theory. *arXiv preprint arXiv:1410.5329*, 2014.

[20] Hosmer D. W. et al. *Applied Logistic Regression*, volume 398. John Wiley & Sons, 2013.

[21] Hsu C., , et al. A Practical Guide to Support Vector Classification. 2003.

[22] Support Vector Machines Tutorial. `https://www.svm-tutorial.com/`. Accessed: 2023-05-20.

[23] Basile V. et al. Semeval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter. In *Proceedings of the 13th international workshop on semantic evaluation*, pages 54–63, 2019.