# Invatare automata - Lucrare practică - Spam Classification

Ignat Gabriel-Andrei
Sorodoc Tudor-Cosmin

12 Ianuarie 2024

---

## 1 Introducere

Pentru a realiza antrenarea si testarea algoritmului nostru de detectare a spamului, am utilizat un set de date compus din 10 foldere, numerorate de la **part1** la **part10**. Aceste foldere sunt impartite in patru categorii principale: **bare, lemm, lemm_stop, stop**.

- **bare** - Lemmatizatorul este dezactivat, lista de oprire este dezactivată.
- **lemm** - Lemmatizatorul este activat, lista de oprire este dezactivată.
- **lemm_stop** - Lemmatizatorul este activat, lista de oprire este activată.
- **stop** - Lemmatizatorul este dezactivat, lista de oprire este activată.

Pentru fiecare folder in parte, pentru antrenare vom folosi folderele **part1** la **part9**, respectiv pentru testare doar folderul **part10**.

Fiecare folder conține un numar de documente text care au fost colectate si **etichetate** anterior. Pentru a intelege continutul acestor documente, am examinat titlurile fisierelor, iar acestea contin un indiciu sub forma prefixului **spm** pentru mesajele de tip spam.
Prin urmare, **etichetele** vor avea valoarea 1 daca denumirea fisierului contine prefixul **spm**, altfel 0.
**Atributele** setului de date includ cuvintele textului conținut în fiecare document.

**Procesul de preprocesare** al datelor include:

- transformarea textului in **lower/upper** case
- inlocuirea numerelor cu un string generic(ex.: **intnumber**)
- inlocuirea adreselor de email cu un string generic(ex.: **emailaddr**)
- inlocuirea link-urilor web cu un string generic(ex.: **httpaddr**)
- inlocuirea simbolurilor valutelor(ex.: $ →**currency_symbol**)
- eliminarea punctuatiei
- eliminarea "stop words"(ex.: **the, of**)
- eliminarea cuvintelor non-alphanumerice(ex.: contin caractere precum {***, @**})
- reducerea cuvintelor la radacina lor sau la forma lor de baza, **normalizarea**(ex.: **registering** si **registered** vor fi aduse la forma **register**).
  Facem acest pas pentru a eliminare redundante/duplicate in setul de atribute de intrare.
  In implementarea noastra putem configura preprocesarea in astfel incat sa folosim 2 astfel de strategii: **lemmatiser**, respectiv **stemmer**(ex.: Snowball Stemmer).

Dupa aceasta preprocesare, vom crea un folder nou **preprocessed_lingspam** in care fiecare email are continut preprocesat anterior.
Totodata, vom pregati terenul si pentru algoritmul propus: pentru fiecare folder **bare, lemm, lemm_stop, stop**, vom construi un dictionar de forma:

{

$$"word1" : [nr_1\_appears\_in\_ham, nr_1\_appears\_in\_spam],$$
$$"word2" : [nr_2\_appears\_in\_ham, nr_2\_appears\_in\_spam],$$
$$\dots$$
$$\}$$

, care reprezinta cuvintele care apar in setul de testare si numarul de mesaje spam in care apare, respectiv mesaje **ham**(i.e. non-spam).

---

# 2 Algoritmul propus

Am ales ca pentru clasificarea email-urilor spam sa folosim algoritmul **Naive Bayes**.
Implementarea lui se gaseste in locatia "**bayes-naive/bayes-naive.py**".

## 2.1 Descrierea Algoritmului

1. **Initializare** - Dupa cum aminteam si mai sus, in timpul preprocesarii se construieste si un dictionar. In momentul instantierii unui clase *Bayes_Naiv*, acesta va primi un dictionar [al carui format este prezentat mai sus]
2. **Antrenare** - In algoritmul Bayes Naiv, antrenarea modelului pe setul de date de antrenare (în cazul nostru, dictionarul primit) nu este un proces complicat. Se calculeaza doar numărul total de aparitii ale cuvintelor în e-mailurile ham și, respectiv, spam.
3. **Testare** - metoda **test** din cadrul clasei, primeste ca argument o lista de teste si returneaza un numaru reprezentand acuratetea. Daca una dintre probabilitati devine zero pentru un cuvant (din cauza lipsei cuvantului intr-una dintre clase în timpul antrenarii ex. : se poate intampla ca "rain" sa apara de nr_1 ori in e-mailurile ham, insa de 0 ori in e-mailurile spam), se activează regula Laplace. Se recalculează probabilitatile cu adaugarea unui factor de smoothing, pentru a evita probabilitati zero. De asemenea, pentru a evita fenomenul de **underflowing**, unde produsul a numeroase probabilități mici poate deveni/(tinde spre) 0, se utilizează logaritmul în calculul acestor probabilități. Acest lucru asigura o reprezentare numerica stabila a rezultatelor.

---

# 3 Justificarea alegerii; comparatie cu alti algoritmi

In contextul clasificarii textuale, unde numarul de atribute poate fi foarte ridicat, algoritmul **Naive Bayes** se comporta mult mai bine fata de alti algoritmi studiati: **ID3, K-nn, AdaBoost**.

De ce am ales **Naive Bayes**?

- **Eficienta computationala**: antrenare in $O(d \cdot n)$, testare in $O(d)$, unde $d$ este numarul de atribute de intrare(numarul de cuvinte din email), iar $n$ este numarul de instante de antrenament.
  Atunci cand numarul de atribute de intrare este foarte mare, ca in cazul clasificarilor textuale, acest aspect devine foarte important.
- **Simplitate in implementare**: **Naive-Bayes** este printre cei mai usor de implementat si de inteles algoritmi de clasificare.
- **Rezistenta la overfitting**: datorita votului majoritar pe care il face, Naive-Bayes reuseste sa evite in cele mai multe cazuri **overfitting-ul**.
- Functioneaza bine cu seturi de **date rare**: in contextul clasificarii textuale, unele cuvinte pot avea o frecventa scazuta. Pot aparea cazuri in care un cuvant nu apartine unei anumite **clase**, astfel probabilitatea calculata de **NB** va fi 0; deci, acel cuvant ar avea o influenta prea mare.

**Din fericire**, Naive-Bayes vine si cu o rezolvare a acestei probleme: **regula lui Laplace "add-one"**.

De ce **NU** am ales ID3?

- **Overfitting**: algoritmul ID3 tinde de a supraspecializa setul de antrenament, avand rezultate slabe in etape de testare pe date noi.
- Crearea **arborilor prea mari**: ID3 are tendinta, mai ales in contextul clasificarii textuale, sa creeze arbori de deicizie prea mari si complecsi.
- **Sensibil la zgomot**: ID3 are tendinta de a se adapta la cele mai mici variatii ale datelor(lucru foarte frecvent in clasificarea textuala), ceea ce in ultima instanta duce la **overfitting.**

De ce **NU** am ales K-nn?

- **Curse of Dimensionality**: K-nn poate intampina dificultati in clasificare atunci setul de date este de dimensiuni ridicate, precum cele din clasificarea textuala.
- Dependenta fata de parametrul $K$: alegerea valorii optime pentru $K$ poate fi o provocare. Un $K$ prea mic poate duce la **sensibilitate la zgomot**, iar un $k$ prea mare poate duce la diluarea informatii importante.
- **Sensibil la zgomot**: punctele **exceptii/zgomot**(outliers) pot influenta semnificativ deciziile lui Knn.

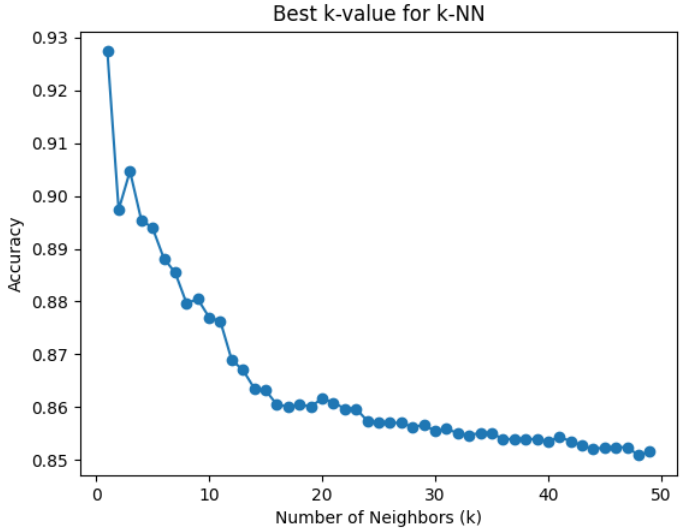
De ce **NU** am ales AdaBoost?

- Si acest algoritm este **sensibil la zgomot**
- Necesita clasificatori slabi **eficienti**: AdaBoost depinde de capacitatea clasificatorilor slabi de a obține performante usor mai bune decat alegerea random. Daca clasificatorii slabi nu sunt suficient de buni, AdaBoost poate avea probleme in construirea unui clasificator puternic.
- **Timp de antrenare mai lung** comparativ cu alti algoritmi, deoarece implica construirea si antrenarea secventiala a mai multor clasificatori slabi.

**Concluzie**

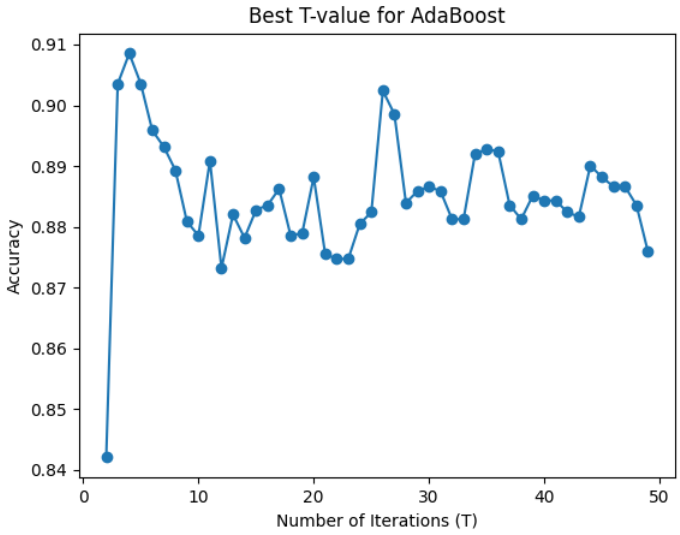
- Luat individual **Naive-Bayes** aduce avantaje importante in clasificarea mesajelor spam: **eficienta** chiar si cand numarul de atribute este semnificativ ridicat, **simplitate** in implementare, **rezistent la zgomot si overfitting**.
- In comparatie cu ceilalti algoritmi, observam ca acestia au niste probleme fundamentale pentru clasificarea mesajelor spam: **problema marilor dimensiuni** la Knn, **overfitting** la ID3, **antrenare costisitoare** si **dependenta fata de clasificatorii slabi** la AdaBoost.

Comparatie **experimentala**.
Pentru **Knn** am ales $K = 1$, deoarece pentru aceasta valoare am obtinut,experimental, cea mai mare acuratete la

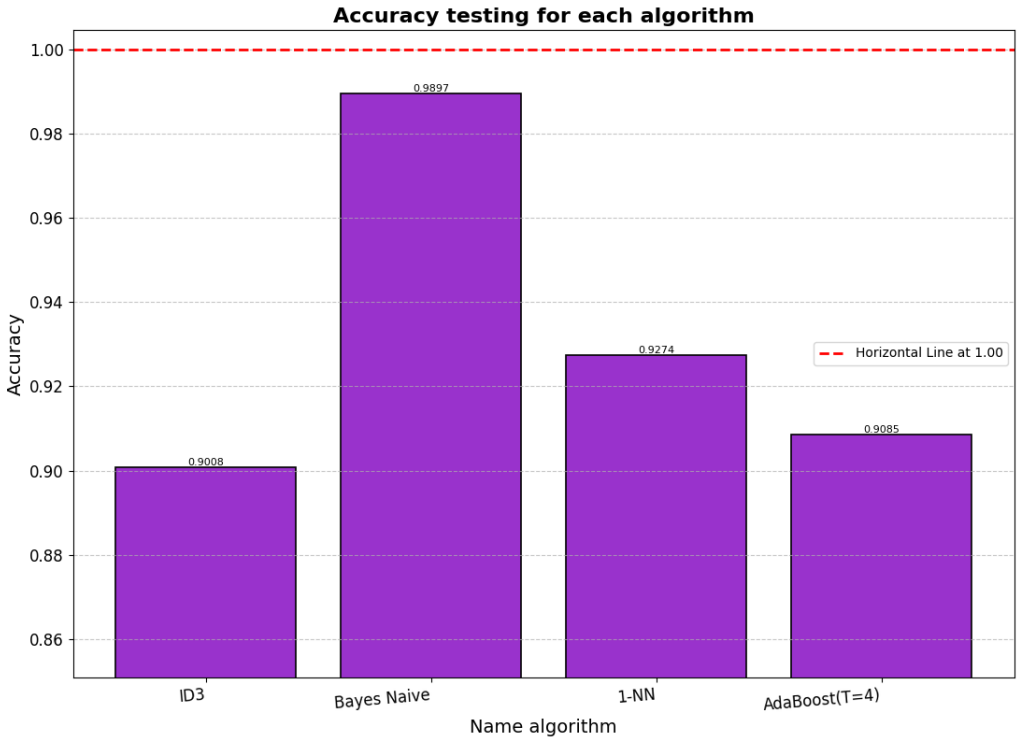



**testare**.

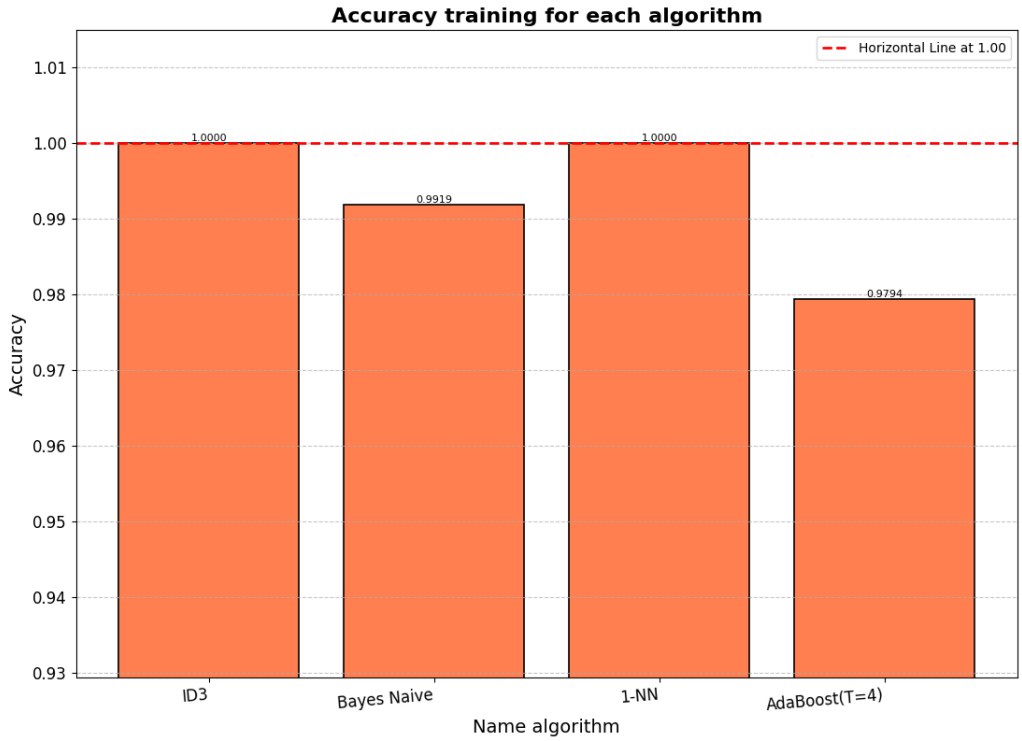
Pentru **AdaBoost** am ales $T = 4$, deoarece pentru aceasta valoare am obtinut,experimental, cea mai mare acuratete la **testare**.




- acuratete la **testare** $\rightarrow$ observam ca **Naive-Bayes** obtine cea mai mare acuratete.




- acuratete la **antrenare**. Desi **ID3**, **Knn** au acuratete mai mare la **antrenare** decat **Naive-Bayes**, observam ca ele sufera de **overfitting**.

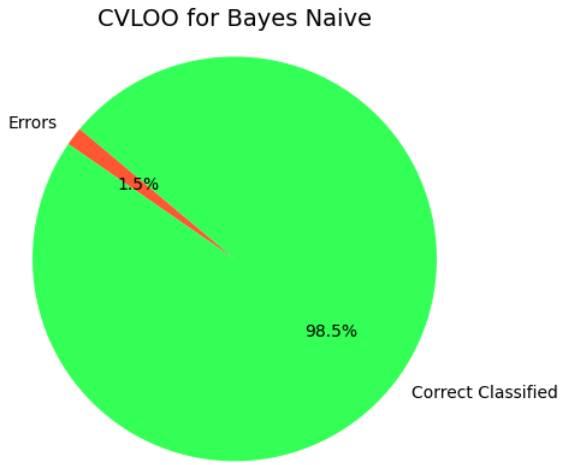



---

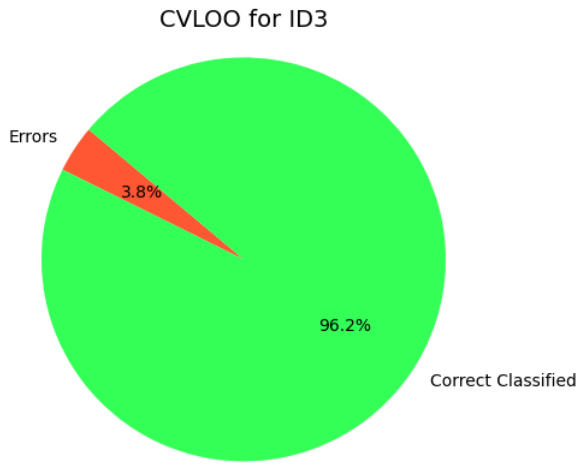
# 4 CVLOO

CVLOO: cross-validation leave-one-out, este o masura de performanta de generalizare a unui algoritm de clasificare. In cele ce urmeaza, vom vedea cum fiecarea algoritm se comporta la CVLOO.

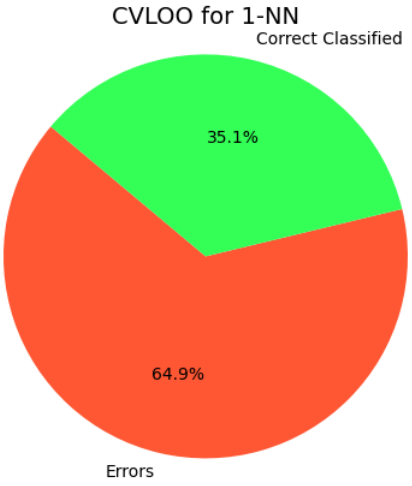
- Naive-Bayes: observam ca si in cazul unor variatii ale setului de antrenare, **NB** inca produce performante de top.



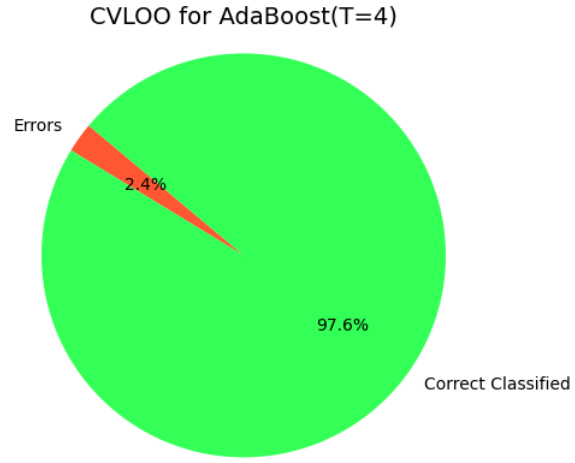

- ID3: observam ca este sensibil la variatii mici a setului de antrenare. Din cauza overfitting-ului, el nu reuseste sa clasifice corect toate instantele eliminate la CVLOO.




- 1-NN: aici observam si un mai mare impact; cauza ar fi ca cel mai apropiat vecin al instantei de clasificat nu mai este acum ea insasi, ci o alta instanta. Prin urmare, putem spune clar ca **K-nn** este sensibil la zgomote si variatii mici a setului de antrenare.



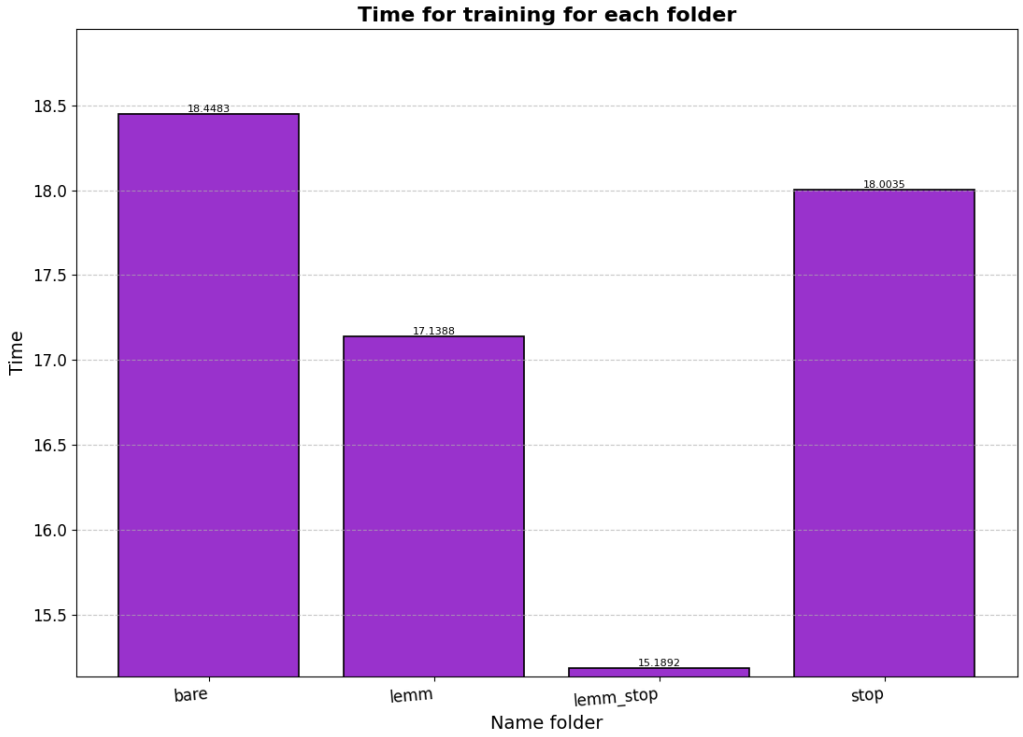

- AdaBoost: performantele raman similare, datorita in mare parte a rezistentei la overfitting.



# 5 Alte aspecte

**Importanta preprocesarii:**

- Dupa cum am metionat si in prima parte, din punct de vedere al preprocesarii am aplicat numeroase metode (eliminarea "stop-words", eliminarea numerelor specifice, eliminarea redundantelor(stemmer, lemmatiser), ..etc). In continuare, vom prezenta 2 grafice in care putem observa diferenta dintre timpul de testare/antrenare atunci cand la partea de preprocesare s-a aplicat doar transformarea textului in lower/upper case, eliminarea punctuatie si eliminarea cuvintelor non-alphanumerice.

Se poate observa ca in medie, timpul de antrenare/testare pentru prima solutie (atunci cand toate preprocesarile au fost aplicate) este mai mic cu aprox. 3.5s.

Prin urmare, procesul de **preprocesare** este foarte important intrucat poate duce si la eficienta d.p.d.v. al timpului de antrenare/testare.