Învățare Automată
Tema Practică
Zolandkovschi Andreea, Cotruță Valeria, grupa 3A1

1 Introducere

Tema presupune analiză, atât dintr-o perspectiva teoretică, cât și printr-o evaluare experimentală, nivelul de potrivire și eficacitate al algoritmilor abordați până în prezent la Învățare Automată în contextul identificării email-urilor spam, folosind setul de date *Ling-Spam*.

Ling-Spam este un director ce conține 4 subdirectoare:

1. *bare*: Lematizatorul și lista de cuvinte comune nu sunt utilizate;
2. *lemm*: Lematizatorul este folosit, dar lista de cuvinte comune nu este utilizată;
3. *lemm_stop*: Lematizatorul și lista de cuvinte comune sunt utilizate;
4. *stop*: Lematizatorul nu este folosit, dar lista de cuvinte comune este utilizată.

Fiecare director conține 10 subdirectoare (part1, part2,..., part10), iar acestea conțîn, la rândul lor, un număr considerabil de fișiere ce includ mesaje de e-mail, fie spam, fie non-spam.

2 Preprocesarea Datelor

```
class Preprocessing:
  def __init__(self, categories):
        self.categories = categories
        self.spam_emails = list()
        self.ham_emails = list()
```

Constructorul primește o lista de categorii, și inițializează două liste goale pentru email-urile spam și ham.

```
def data_split(self, folder_path, train_data_folder, test_data_folder)
```

Această metodă separă în două foldere datele de antrenare și datele de testare. Fișierele din 'part10' vor fi puse în folderul de testare, restul fișierelor fiind puse în folderul de antrenare. La final email-urile vor fi puse pe categorii.

```
def train_labels(self, train_data_folder)
```

Construiește dicționare separate pentru cuvintele din email-urile ham și spam în datele de antrenament, folosind modulele Counter și re.

```
if is_spam:
    if category not în word_dictionary_spam:
       word_dictionary_spam[category] = Counter()
    word_dictionary_spam[category] += Counter(words)
else:
   if category not în word_dictionary_ham:
          word_dictionary_ham[category] = Counter()
   word_dictionary_ham[category] += Counter(words)
```

Cuvintele sunt filtrate să conțină doar litere.

3 Justificare Teoretică și Experimentală

Drept algoritm de clasificare a email-urilor, am selectat algoritmul Bayes Naiv, întrucât este adesea preferat pentru problemele de clasificare a textului, cum ar fi identificarea mesajelor spam.

Bayes Naiv funcționează bine la manipularea datelor textuale și facilitează înțelegerea sporită a frecvenței acestora, este robust pe seturi de date de dimensiuni mici și moderate. Comparând algoritmul Bayes Naiv cu arborii de decizie:

1. Ambii reprezintă algoritmi relativ minimalisti și ușor de implementat.

2. Ambii algoritmi operează bine cu date care au caracteristici simple sau caractere independente, în cazul nostru, cuvinte.

Totuși, în detrimentul algoritmului bazat pe arborii de decizie care construiesc un set de reguli de decizie pentru a clasifică datele, la baza algoritmului Bayes Naiv se află teorema lui Bayes care presupune independența între caracteristici. De exemplu, apariția cuvântului "funcționează" într-un e-mail nu este influențată de prezența altor cuvinte-cheie, precum "merge" sau "lucrează", ceea ce s-ar putea întâmplă în celălalt algoritm.

Arborii de decizie pot avea tendința de overfitting pe seturi de date uriașe sau cu un număr considerabil de parametri parametri, în timp ce Naive Bayes poate fi mai puțîn sensibil la supradimensionare.

Adițional, Bayes Naiv poate rula bine pe seturi de date mari, pe când arborii de decizie pot deveni ineficienți.

În continuare, vom explică modul de implementare a algoritmului Bayes Naiv.

3.1 Bayes Naiv

```
class NaiveBayesClassifier:
    def __init__(self, spam_emails_words_dictionary,
                 ham_emails_words_dictionary, categories):
        self.spam_words_dictionary = spam_emails_words_dictionary
        self.ham_words_dictionary = ham_emails_words_dictionary
        self.categories = categories
```

Constructorul inițializează dicționarele de cuvinte pentru email-uri spam și email-uri ham, precum și categoriile.

```
def classify(self, file_path, spam_emails_number_category,
             ham_emails_number_category, words=[]):
```

Metodă clasifică un fișier dat în spam sau ham pe baza probabilităților calculate în metodă:

```
@staticmethod
def compute_probability(file_words, word_dictionary,
        total_emails_number, probability):
```

3.2 Acuratețea algoritmului NB pe setul de date de antrenare

Probabilitățile inițiale în algoritmul Naive Bayes pot influența semnificativ performanța acestuia. Deoarece setul nostru de date conține o disproporție semnificativă între numărul de exemple de spam și ham, aceste probabilități inițiale pot avea un impact mare.

De aceea, pentru a evita depășirea limitei valorice a tipului de date "float" și asignarea instanța 0.0 probabilităților condiționate ce urmează a fi calculate pentru fiecare cuvânt din e-mail, am utilizat următoarele formule:

```
ham_probability = math.log(spam_emails_number_category / (
        spam_emails_number_category + ham_emails_number_category))
spam_probability = math.log(ham_emails_number_category / (
        spam_emails_number_category + ham_emails_number_category))
```

Mai mult de atât, utilizarea logaritmului pentru calculul probabilităților condiționate poate ajută la evitarea problemelor legate de precizia numerelor în virgulă mobilă în Python, mai ales când probabilitățile sunt foarte mici, însă sumele de logaritmi pot rezultă în aplicarea funcției logaritm peste numere negative. Pentru a soluționa această problema, am condiționat calcularea probabilităților prin

```
  if word_dictionary[word] > 0 and total_emails_number + vocabulary_size > 0:
  if total_emails_number + vocabulary_size > 0:
 din metodă statică
def compute_probability(file_words, word_dictionary, total_emails_number, probability)
```

. În acest mod, am obținut o acuratețe destul de bună în ceea ce privește clasificarea exemplelor din setul de testare propus. Acuratețea algoritmului Bayes Naiv pe setul de date de testare: 99.65635738831615

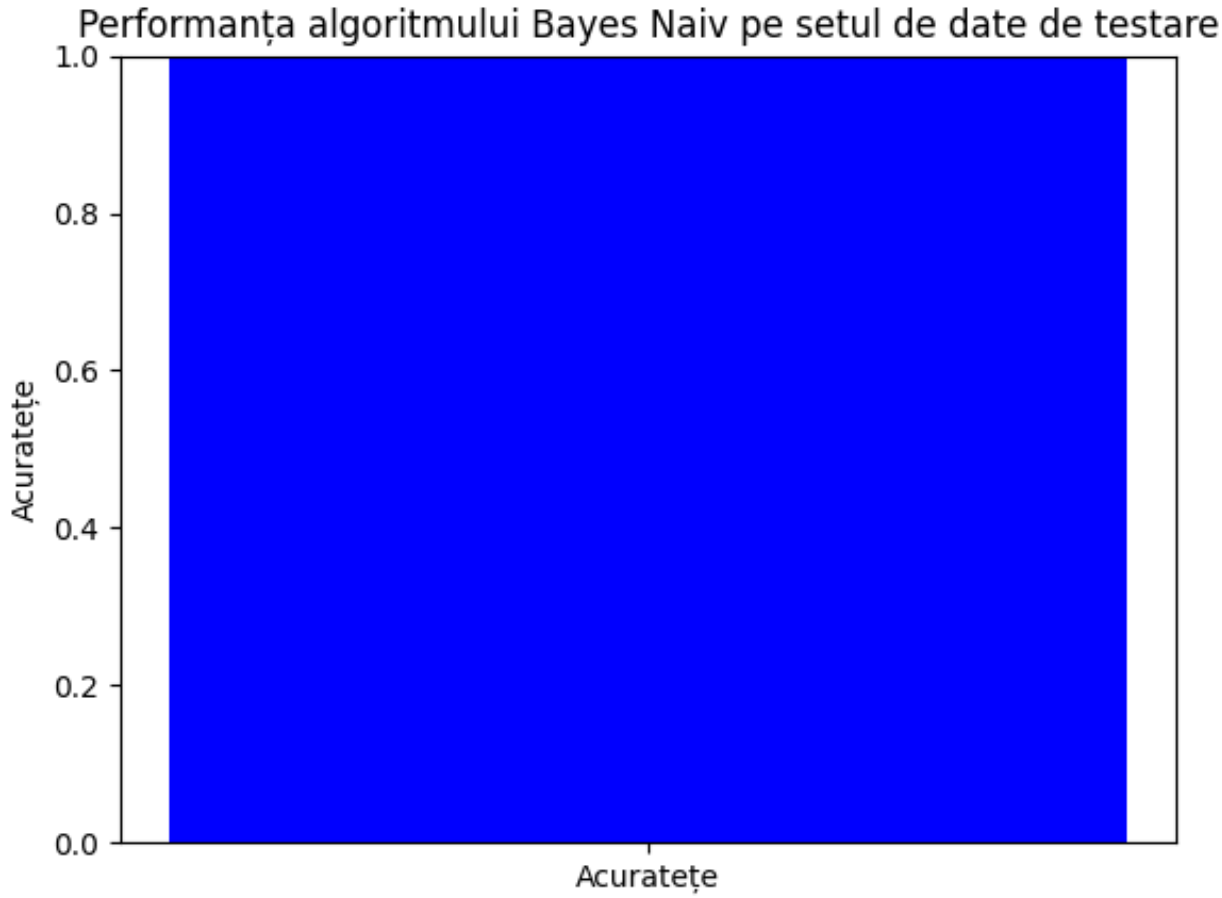

Figure 1: Acuratețe Bayes Naiv

3.3 Leave-One-Out Cross-Validation

În cadrul clasei, se parcurge directorul de date de antrenare recursiv pentru a identifica toate fișierele de tip .txt. Pentru fiecare email, se extrag cuvintele, iar apoi fiecare cuvânt este eliminat secvențial. După acest pas, dicționarele asociate cu e-mailuri spam și ham sunt actualizate prin eliminarea cuvintelor din setul curent.

Următorul pas constă în crearea unei instanțe a clasificatorului Naive Bayes cu dicționarele actualizate și categoriile corespunzătoare. Se obțîn numerele totale de e-mailuri spam și ham pentru fiecare categorie, iar apoi se efectuează clasificarea fiecărui e-mail în parte. Eticheta clasificată este comparată cu cea reală pentru a calcula acuratețea, iar acest rezultat este înregistrat într-o listă.

La finalul procesului pentru toate fișierele, obiectul este reinițializat pentru a pregăti următoarea iterație. Procentajul mediu de acuratețe este calculat și returnat că rezultat într-un grafic(Figure 2).

3.4 Distribuția Accurateții Leave-One-Out Cross-Validation

Leave-One-Out este o modalitate repetitivă, eficientă pentru validare în care, pentru fiecare exemplut din setul de date, se antrenează algoritmul pe toate celelalte exemple și se evaluează pe exemplul extras. Acest proces, pentru a obține o acuratețe mai exactă, se repetă de câteva ori, apoi se calculează performanță medie.

Distribuția accurateții Leave-One-Out Cross-Validation după 10 rulări este prezentată în lista următoare:

```
[63.91237509607994, 99.86548808608762, 99.86548808608762, 99.86548808608762,
99.86548808608762, 99.86548808608762, 99.86548808608762, 99.86548808608762,
99.86548808608762, 99.86548808608762]
```

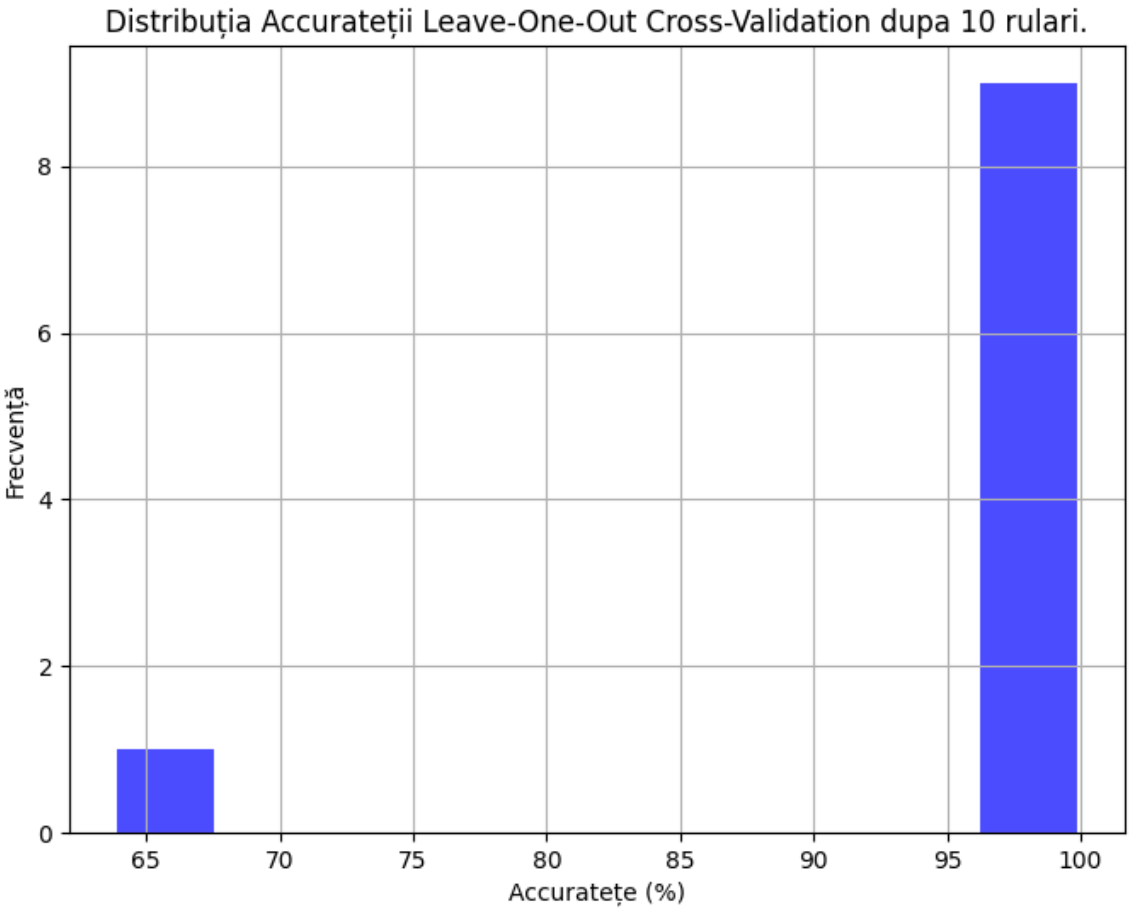

Figure 2: Acuratețe CVLOO

3.5 Acuratețea algoritmului pe setul de date de antrenare

Acuratețea algoritmului Leave-One-Out Cross-Validation pe setul de date de antrenare este de 96.27017678708685.%

4 Concluzii

Am studiat și implementat un algoritm bazat pe Bayes Naiv pentru detectarea mesajelor de spam, evaluându-l pe setul de date Ling-Spam. Procesul de pregătire a datelor a fost esențial, implicând tehnici precum împărțirea în seturi de antrenament și testare, construirea de dicționare pentru cuvintele cheie și eliminarea cuvintelor inutile.

Algoritmul Bayes Naiv a obținut o precizie de aproximativ 99.66% în identificarea mesajelor de spam pe setul de date de testare, validându-și capacitatea de generalizare și clasificare eficientă a spamului și a mesajelor normale.

Folosind Leave-One-Out Cross Validation, am evaluat performanța algoritmului pe setul de date de antrenament, obținând o acuratețe de aproximativ 99.66%. Aceasta sugerează o bună capacitate de adaptare a algoritmului la date noi.

Comparând Bayes Naiv cu arborii de decizie, am identificat avantaje și dezavantaje specifice fiecărui algoritm. Bayes Naiv s-a dovedit robust în manipularea datelor textuale, având o performanță semnificativă în identificarea mesajelor de spam spre deosebire de arborii de decizie care pot suferi de overfitting la seturi mari de date sau seturi cu mulți parametri.

În concluzie, algoritmul Bayes Naiv este o alegere adecvată pentru detectarea email-urilor spam sau non-spam.