

Tema practica

Socoteală Andrei Alin, Adăscăliței Robert Marian

Jan 2024

Întelegerea setului de date

Setul de date conține 4 foldere (lemm, bare, lemm_stop și stop), iar fiecare folder conține alte 10 foldere care conțin email-uri (part1-part10). Mesajele de tip spam conțin în titlu cuvântul "spm". Pentru rezolvare, am folosit folderul bare, iar aici am utilizat primele 9 foldere pentru antrenare, iar folderul 10 pentru testare. Pentru procesarea datelor, am folosit o metodă simplă: am utilizat un dicționar care conține conținutul email-ului și eticheta "spam" (dacă mesajul este spam) sau eticheta "regular" (dacă mesajul nu este spam). Clasificarea în email spam sau email regular se face după numele fișierului (dacă conține "spm" este spam, altfel regular). Funcția care clasifică email-urile poate fi văzută mai jos.

```
def load_emails(directory):
    emails = []
    for filename in os.listdir(directory):
        if filename.startswith("spm"):
            with open(os.path.join(directory, filename), 'r', encoding='latin-1') as file:
                content = file.read()
                emails.append((content, 'spam'))
        else:
            with open(os.path.join(directory, filename), 'r', encoding='latin-1') as file:
                content = file.read()
                emails.append((content, 'regular'))
    return emails
```

În continuare, pentru partea de antrenare, se extrage fiecare cuvânt din fiecare email și crește count-ul (count-ul pentru spam dacă a fost găsit într-un email spam sau count-ul pentru regular dacă a fost găsit într-un email regular). În final, după faza de antrenare, vom avea două dicționare, unul

pentru cuvinte spam, unul pentru cuvinte regular care vor contine cuvinte si respectiv count-ul lor. Mai jos, functia tokenize si functia de antrenare:

```
def tokenize(email):
    return [word.strip(",.") for word in email.split()]

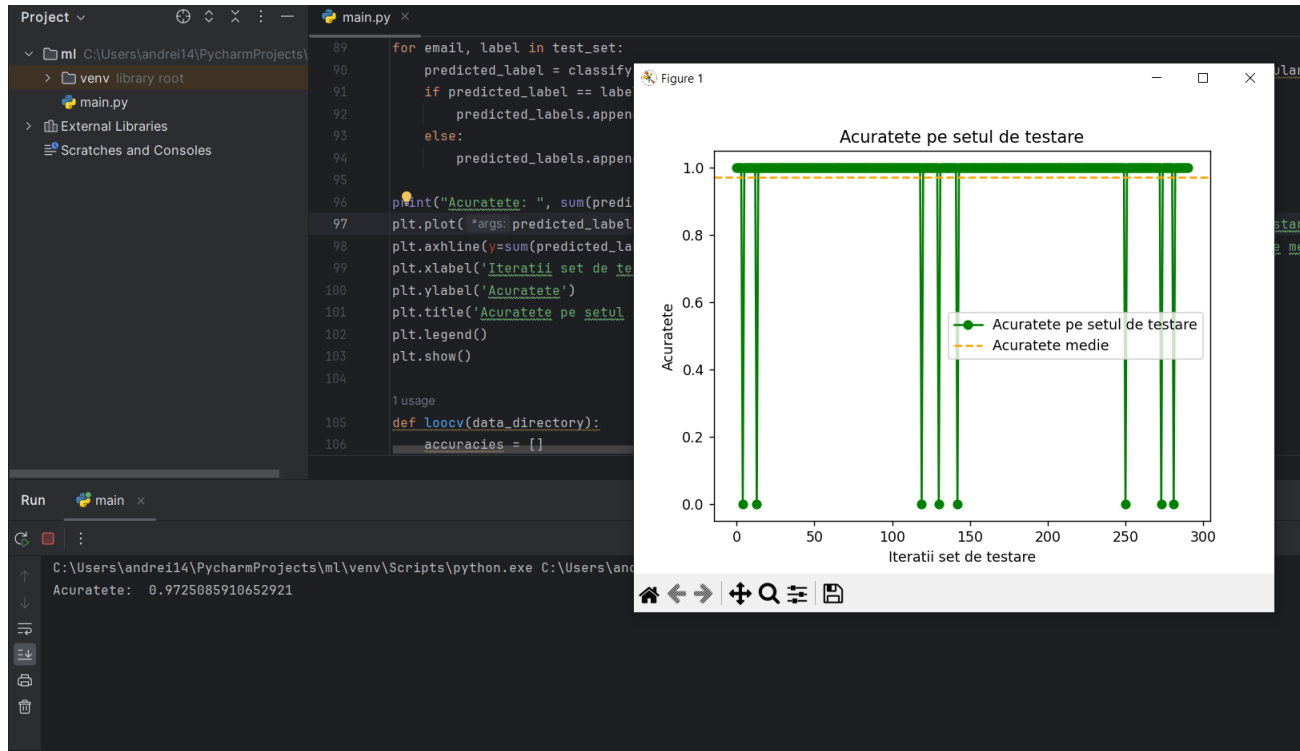
2 usages
def train_naive_bayes(train_set):
    #la antrenare, calculam frecventa cuvintelor din fiecare clasa
    spam_word_counts = defaultdict(int)
    regular_word_counts = defaultdict(int)
    spam_total_words = 0
    regular_total_words = 0
    spam_email_count = 0
    regular_email_count = 0

    for email, label in train_set:
        words = tokenize(email)
        if label == 'spam':
            spam_email_count += 1
            for word in words:
                spam_word_counts[word] += 1
            spam_total_words += 1
        else:
            regular_email_count += 1
            for word in words:
                regular_word_counts[word] += 1
            regular_total_words += 1

    return spam_word_counts, regular_word_counts, spam_total_words, regular_total_words, spam_email_count, regular_email_count
```

Alegerea algoritmului

Pentru problema clasificarii email-urilor spam din setul de date Ling-Spam am ales algoritmul Naive Bayes, performantele acestuia pot fi vazute, in imaginea ce urmeaza, atat grafic cat si text:



Justificare teoretica:

Naive Bayes se bazează pe teorema lui Bayes și are presupozitia că toate caracteristicile sunt independente între ele, date fiind clasa. În contextul clasificării email-urilor spam, această abordare simplificată se potrivește bine, deoarece email-urile pot avea un număr mare de cuvinte și caractere, iar presupozitia de independență poate facilita calculul probabilităților condiționate. De asemenea, Naive Bayes este eficient în gestionarea datelor rare și este rapid de antrenat, ceea ce îl face potrivit pentru seturi de date de dimensiuni mari, cum ar fi colecțiile de email-uri. Naive Bayes este, de asemenea, adesea folosit pentru clasificări bazate pe text (exact ca și în cazul email-urilor).

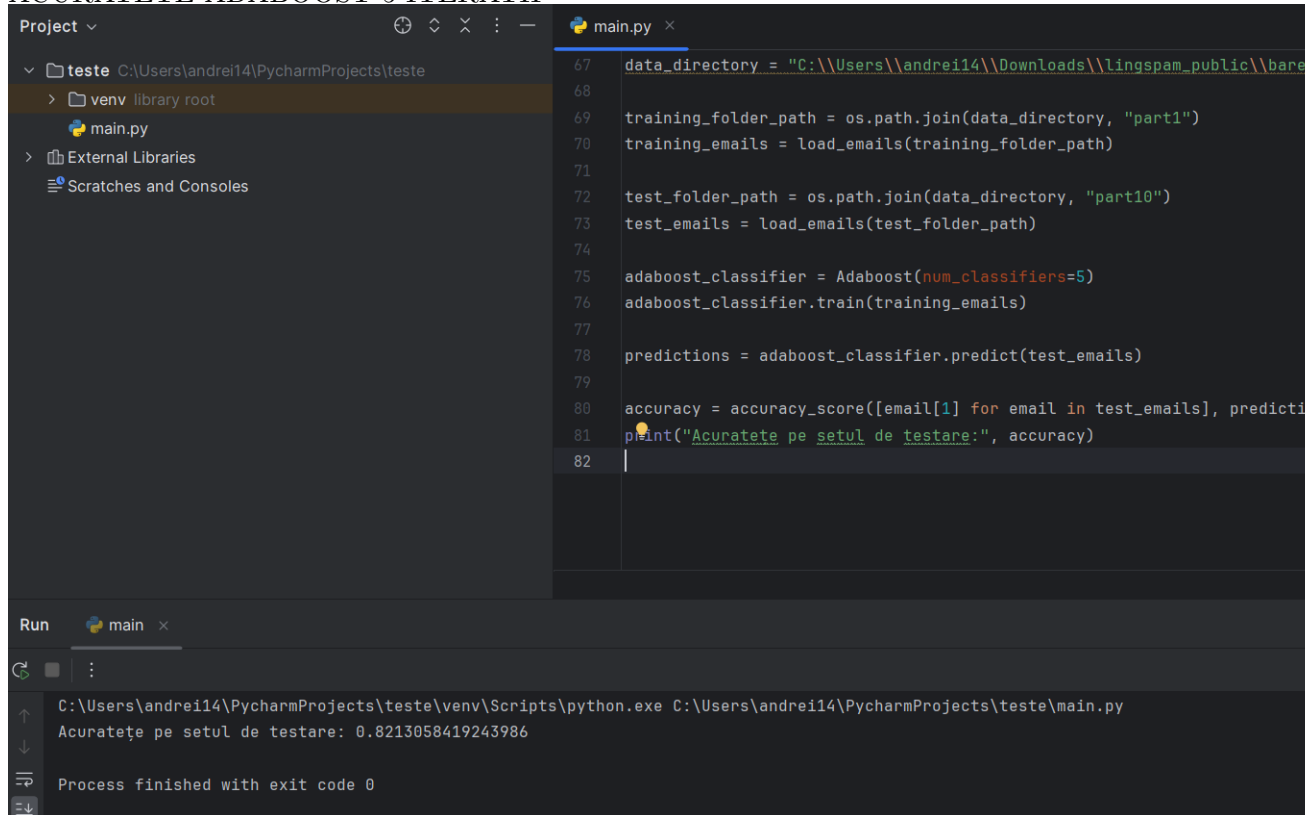
Din acest motiv am ales Naive Bayes și nu, de exemplu ID3, unde, având un set de date foarte complex, performanța putea fi afectată (de exemplu, din cauza overfitting-ului).

Justificare practica:

Mai sus am arătat acurătatea pe setul de testare pentru algoritmul Bayes Naiv, care este una de aproximativ 97%, o acurătate mult mai bună de-

cat o acuratete pentru datul cu banul sau alegerea aceleiasi clase intotdeauna (daca alegeam intotdeauna clasa regular, am fi avut acuratete de 83%, deoarece in bare-part10 sunt 242 mesaje regular si 49 mesaje spam). In plus, am testat acelasi set de date si pentru un clasificator de tipul Adaboost, folosind 5 iteratii, acuratetea in acest caz fiind de 82%, adica aproximativ cat cea de la alegerea clasei regular intotdeauna.

ACURATETE ADABOOST 5 ITERATII



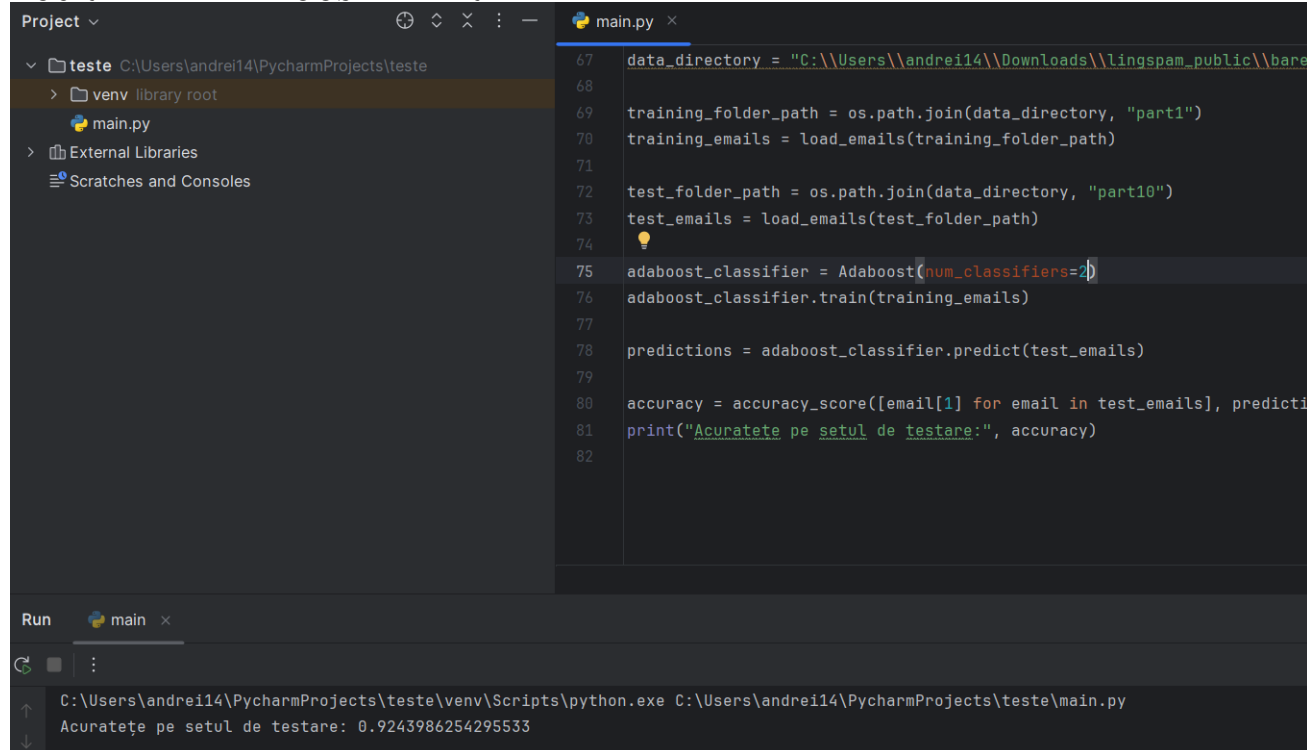
```
Project
└─ teste C:\Users\andrei14\PycharmProjects\teste
    └─ venv library root
        └─ main.py
    └─ External Libraries
    └─ Scratches and Consoles

main.py
67 data_directory = "C:\\Users\\andrei14\\Downloads\\lingspam_public\\bare
68
69 training_folder_path = os.path.join(data_directory, "part1")
70 training_emails = load_emails(training_folder_path)
71
72 test_folder_path = os.path.join(data_directory, "part10")
73 test_emails = load_emails(test_folder_path)
74
75 adaboost_classifier = Adaboost(num_classifiers=5)
76 adaboost_classifier.train(training_emails)
77
78 predictions = adaboost_classifier.predict(test_emails)
79
80 accuracy = accuracy_score([email[1] for email in test_emails], predictions)
81 print("Acuratete pe setul de testare:", accuracy)
82

Run
main
C:\Users\andrei14\PycharmProjects\teste\venv\Scripts\python.exe C:\Users\andrei14\PycharmProjects\teste\main.py
Acuratete pe setul de testare: 0.8213058419243986
Process finished with exit code 0
```

Pentru un clasificator de tipul Adaboost dar doar cu 2 iteratii, am obtinut o acuratete de 92%, mult mai buna decat cea anterioara, dar totusi sub cea obtinuta cu Naive Bayes.

ACURATETE ADABOOST 2 ITERATII



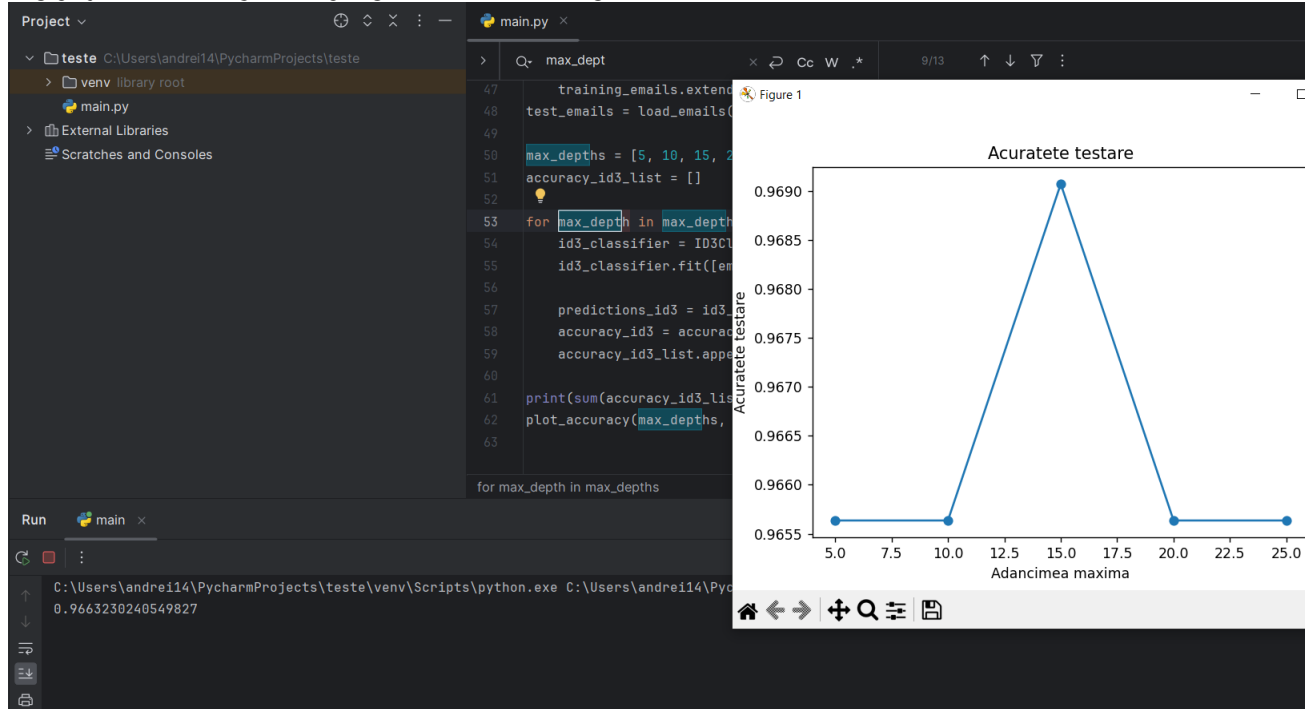
```
Project ▾
  ▾ teste C:\Users\andrei14\PycharmProjects\teste
    > ▾ venv library root
      main.py
    > ▾ External Libraries
      Scratches and Consoles

main.py ×
67 data_directory = "C:\\Users\\andrei14\\Downloads\\lingspam_public\\bare
68
69 training_folder_path = os.path.join(data_directory, "part1")
70 training_emails = load_emails(training_folder_path)
71
72 test_folder_path = os.path.join(data_directory, "part10")
73 test_emails = load_emails(test_folder_path)
74
75 adaboost_classifier = Adaboost(num_classifiers=2)
76 adaboost_classifier.train(training_emails)
77
78 predictions = adaboost_classifier.predict(test_emails)
79
80 accuracy = accuracy_score([email[1] for email in test_emails], predicti
81 print("Acuratete pe setul de testare:", accuracy)
82

Run main ×
C:\Users\andrei14\PycharmProjects\teste\venv\Scripts\python.exe C:\Users\andrei14\PycharmProjects\teste\main.py
Acuratete pe setul de testare: 0.9243986254295533
```

Mai jos, acuratetea pentru algoritmul ID3, una de aproximativ 96.6%, este cea care se apropie cel mai mult de cea obtinuta de Naive Bayes, dar are in continuare un procent care poate fi semnificativ sub Naive Bayes.

ACURATETE ID3 IN FUNCTIE DE ADANCIMEA MAXIMA



Cele 2 implementari pentru Adaboost si ID3 au fost implementari simple folosind notiunile deja existente din sklearn si nu implementari complete cum este cea pentru Naive Bayes.

Pe baza rezultatelor practice, dar si a informatiilor teoretice, am observat ca Naive Bayes obtine o performanta superioara pentru acest tip de problema fata de ceilalti algoritmi studiati si este de asemenea si mai usor si mai intuitiv de implementat.

LOOCV

Dupa cum se poate observa mai jos, acuratetea LOOCV este aproximativ 97%, aproximativ egala cu acuratetea pe setul de date de test. Acest lucru arata ca modelul implementat are o buna putere de generalizare pe date noi si de asemenea, lipseste overfitting-ul.

