Vîlcu Rareș-Andrei - Clasificator Naive Bayes pentru Emailuri Spam/Ham

1. Introducere

În acest proiect am implementat un clasificator simplu de tip Bayes multinomial pentru texte, cu scopul de a distinge între emailuri de tip spam și emailuri normale (ham). Modelul e făcut în Python cu structuri de date de bază și concepte matematice simple (probabilități, logaritmi).

2. Setul de date

Pentru proiect am folosit un set de date construit de mine, plecând de la emailuri reale
Le-am pus într-un fișier emails.csv în format:

text,label

"Text email 1",ham

"Text email 2",spam

Textele sunt procesate cu o funcție tokenize care elimină punctuația și împarte textul în cuvinte.

3. Modelul Naive Bayes

Modelul folosește o idee simplă: pentru orice text, vrem să calculăm ce clasă este mai probabilă:

$$P(c|text)$$

unde c poate fi spam sau ham.

Prin teorema lui Bayes, asta devine:

$$P(c|text) = \frac{P(text|c) * P(c)}{P(text)}$$

În practică nu ne interesează P(text) pentru că este același pentru toate clasele, deci putem compara doar:

$$P(text|c) * P(c)$$

P(c) = probabilitatea ca un email oarecare să fie din clasa c.
O calculăm numărând câte emailuri spam/ham avem în dataset.

$$P(c) = \frac{nr\ emailuri\ in\ clasa c}{nr\ total\ emailuri}$$

P(text | c) = probabilitatea de a vedea cuvintele din text dacă clasa este c.
Pentru asta presupunem că fiecare cuvânt e independent („naive").

Modelul devine:

$$P(text|c) = \prod_{w \epsilon text} P(w|c)$$

Pentru P(w | c) folosim Laplace smoothing:

$$P(w|c) = \frac{(count(w,c) + 1}{totalcuvinteinclasa + |V|}$$

4. Implementarea în Python

Structura codului

- tokenize: curăță textul

- load_dataset: citește CSV

- train_test_split: împarte datele

- BayesMultinomial: conține logica modelului

- accuracy_score: calculează acuratețea

- main

Principalele funcții sunt:

- fit(texts, labels)

învață din date:

calculează P(c) – probabilitățile a priori,

numără aparițiile fiecărui cuvânt în fiecare clasă,

salvează vocabularul și numărul total de cuvinte.

- word_prob(word, c)

calculează P(w | c) cu Laplace smoothing.

- predict_one(text)

aplică formula logaritmată:

$$logP(c) + \sum_{i=1}^{n} (logP(w_i|c))$$

întoarce clasa cu cel mai mare scor.

- predict(texts)

aplică predict_one pe fiecare text din listă.

5. Clasificarea textelor noi

Pentru un text nou calculăm:

$$logP(c) + \sum_{i=1}^{n} (logP(w_i|c))$$

și alegem clasa cu scorul cel mai mare.

6. Cum rulezi programul

- Pentru rulare:

python bayes.py

- sau, dacă vrei să specifici alt fișier CSV:

python bayes.py alt_dataset.csv

Programul:

încarcă fișierul,îl împarte în 70% train, 30% test, antrenează modelul, evaluează acuratețea, afișează câteva exemple, deschide un mod interactiv, unde poți scrie tu un text.

- Exemplu de test:

input:

> Felicitări, ai câștigat un premiu garantat!

prediction: spam

7. Acuratețea obținută variază în funcție de dataset. În cazul meu, cu emailuri reale, am obținut în jur de **75–90%**

Modelul se descurcă bine mai ales cu cuvinte distinctive precum „premiu”, „urgent”, „moștenire”, „factură”, „abonament”, „confirmare”, etc.

8. Bibliografie

https://www.youtube.com/watch?v=jSaU_iDB1Ds

Wikipedia – Naive Bayes classifier