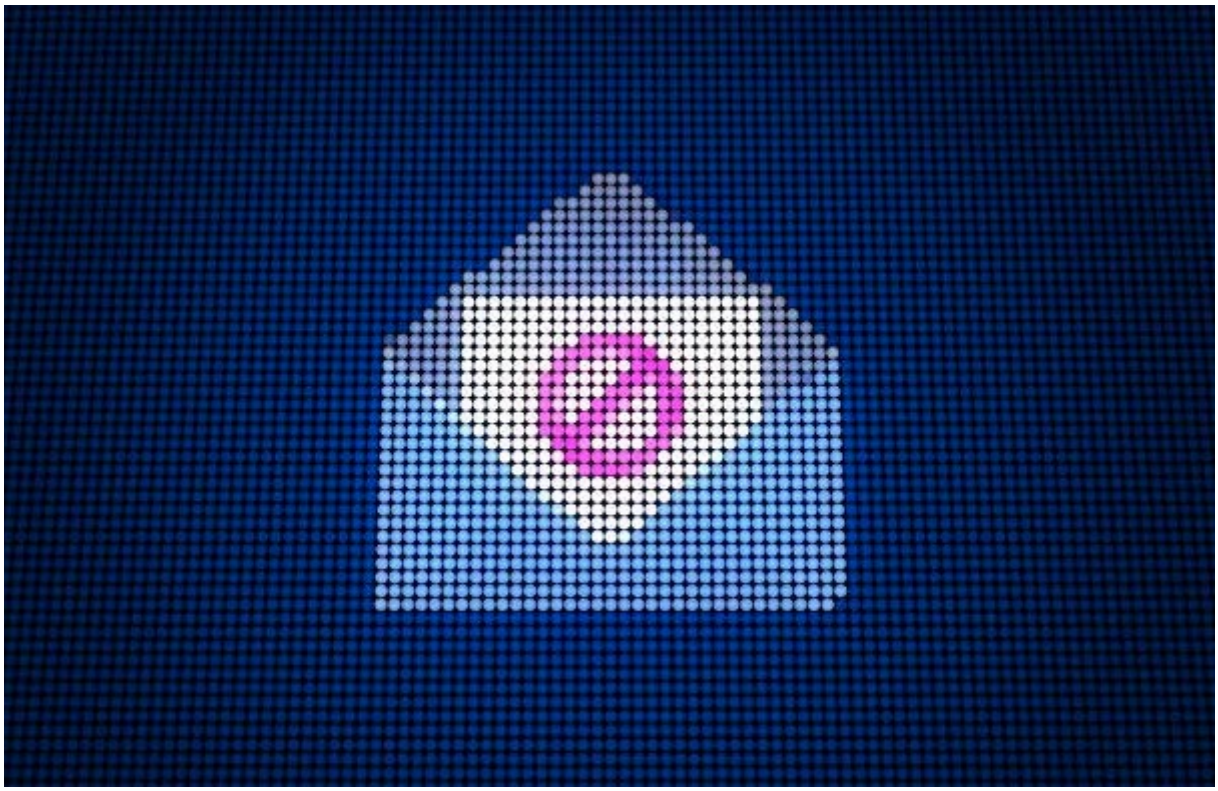


1. Introducere

Spamul este una din cele mai răspândite categorii de mail. Mesajele nesolicitate pot duce la productivitate scăzută, pot cauza discomfort sau chiar breșe de securitate. Algoritmii ML au devenit standardul în automatizarea detecției acestor mesaje. În acest proiect vom antrena cateva modele și vom urma procesul KDD pentru a selecta, curăța, procesa și modela setul de date Spambase pentru clasificarea spamului.
Pentru realizarea acestui proiect s-au folosit în întregime Python 3.10.17 și ecosistemul scikit-learn.

Intrebare de cercetare: “Cât de bine putem clasifica mailuri ca spam sau non-spam doar pe baza atributelor extrase din textul mesajului.”

2. Selecția Datelor

2.1 Justificarea setului de date

De ce spambase? – este gata procesat, nu are valori lipsă, are doar atribute numerice și conține destule date cât să antrezi niste modele la scara proiectului nostru. Conține date reale și este foarte relevant pentru clasificare binară.

Am importat setul de date folosind OpenML și am atașat fisierul CSV in arhiva de pe moodle pentru vizibilitate și accesibilitate.

```
#Am folosit OpenML pentru a elimina folosirea fisierelor locale, executare usoara, reproductivitate si portabilitate mai mare.
#Csv-ul spambase e inclus in arhiva de pe moodle
df = fetch_openml(name="spambase", version=1, as_frame=True).frame
```

Screenshot: încărcarea setului de date folosind OpenML.

2.2 Descrierea caracteristicilor și variabilei țintă

Fiecare element din Spambase reprezintă un mail, descris de 57 atribute numerice și o coloană țintă numită „class”. Atributele sunt bazate pe frecvența anumitor cuvinte specifice spamului precum „free”, „money”, „you”; Prezența unor caractere anume ca și „!”, „$”, „?”; Lungimea celui mai lung string uppercase.
Variabila țintă „class” e categorică și are valoarea 1 pentru spam și 0 pentru non-spam, prezența ei facilitează aplicarea clasificării binare.

```
   capital_run_length_average  capital_run_length_longest  \
0                       3.756                          61
1                       5.114                         101
2                       9.821                         485
3                       3.537                          40
4                       3.537                          40

   capital_run_length_total  class
0                       278      1
1                      1028      1
2                      2259      1
3                       191      1
4                       191      1

[5 rows x 58 columns]
```

Screenshot: mic extras al atributelor și țintei, folosind print(df.head())

3. Preprocesarea Datelor

3.1 Gestionarea valorilor lipsă

Chiar dacă Spambase este foarte complet și bine închegat, am verificat valorile lipsă pentru a fi siguri. Am verificat întregul dataframe folosind .isnull().sum().sum(). Dacă există valori lipsă, se renunță la ele. Nu s-a inregistrat nici o valoare lipsă.

```
if df.isnull().sum().sum() > 0:
    print("valori lipsa gasite, se elimina...")
    df = df.dropna()
else:
    print("0 valori lipsa, continua rularea\n")
#verificam valori lipsa
```

Screenshot: Verificarea valorilor lipsă și eliminarea lor (dacă există)

3.2 Împărţirea datelor în seturi de antrenare şi testare

For fair and unbiased model evaluation, the dataset was randomly split into training and test sets. I used a 70% training and 30% test split, preserving the original spam/non-spam ratio (stratified split) to ensure that both sets represent the underlying distribution of the data.
This approach enables us to train models and evaluate their generalization on previously unseen data.

Pentru o evaluare cât mai corectă, am împărțit setul de date în training și test. S-a ales un split stratificat cu o rație de 70%-30% pentru a asigura consistenta. În acest fel proporția dintre spam si non-spam este aceeasi la ambele subseturi, tot cât în setul întreg. Random state 42, tot pentru asigurarea consistenței.

```
X = df.drop("class", axis=1)
y = df["class"].astype(int)

date_train, date_test, etichete_train, etichete_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

Screenshot: împărțire în train-test

3.3 Standardizarea caracteristicilor

Atributele din Spambase au scale diferite, unele măsoară frecvența cuvintelor și variază de la 0 la 20, altele numără caracterele uppercase sau simboluri speciale. Modele precum regresia logistică și Gradient boosting sunt sensibile la scala atributelor, deci toti predictorii au fost standardizați folosind StandardScaler din scikit-learn. Scalarea a fost ajustată doar pe train și apoi aplicată pe ambele seturi pentru a evita scurgerea datelor și overfittingul.

```
scaler = StandardScaler()
date_train = scaler.fit_transform(date_train)
date_test = scaler.transform(date_test)
```

Screenshot:standardizarea datelor cu StandardScaler.

4. Transformarea Datelor

Datorită robusteții setului Spambase, nu au fost necesare nici o transformare complexă. Cea mai mare transformare a fost scalarea menționată anterior.

5. Selecția și Antrenarea Modelului

5.1 Alegerea algoritmilor

Am ales 4 strategii de clasificare comune, acestea acoperă atât liniar vs non-liniar dar și individual vs ansamblu. Arborele de decizie crează o singura structură, în timp ce Random Forest se foloseste de mai multe structuri independente. Gradient boosting crează recursiv arbori, corectând erorile individuale. Am ales și Regresia logistică, pentru a avea un model ce nu functionează pe bază de arbori.

- **Regresie Logistică** – model liniar de bază, interpretabil, potrivit pentru date scalate.
- **Arbore de Decizie** – model non-liniar, poate inregistra relații complexe, dar mai predispus la overfitting, mai ales pe seturi cu multe features ca acesta.
- **Random Forest** – ansamblu de arbori slabi, robustețe crescută prin medierea răspunsurilor.
- **Gradient Boosting** – boosting bazat pe arbori, corectare iterativă a erorilor.

5.2 Configurarea și antrenarea modelelor

Fiecare model a fost definit cu hyperparametri default din scikit-learn, cu mici modificări, pentru a măsura performanța „out of the box”, fară prea multă intervenție din partea noastră.

Toate modelele pleacă cu același seed, prin random_state=42.

Am limitat adăncimea arborilor din Decision Tree si Random forest pentru a evita overfitting, am setat numărul de estimatori din Random Forest penru echilibru intre generalizare și timpi de antrenare. La Gradient Boosting am folosit 150 de estimatori și rată de învățare de 0.1 pentru antrenament rapid și stabil.La Regresia Logistică am crescut numărul de iterații la 200 pentru a asigura rularea completă a algoritmului, având in vedere numărul mare de features.

```python
#2 definirea modelelor

modele = {
    "Regresie Logistica": LogisticRegression(max_iter=200, random_state=42),
    "Arbore de Decizie": DecisionTreeClassifier(max_depth=10, min_samples_split=5, random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=150, learning_rate=0.1, random_state=42)
}
```

Screenshot: Definirea modelelor și hyperparametrii lor

```python
#3 Antrenare si evaluare modele

rezultate = []
predictii = {}
probabilitati = {}
matrici_confuzie = {}

for nume, model in modele.items():
    print(f"Antrenare si evaluare model: {nume}")
    model.fit(date_train, etichete_train)
    pred_train = model.predict(date_train)
    pred_test = model.predict(date_test)
    prob_test = model.predict_proba(date_test)[:, 1]
    scor_auc = roc_auc_score(etichete_test, prob_test)

    rezultate.append({
        "Model": nume,
        "Acuratete Antrenare": accuracy_score(etichete_train, pred_train),
        "Acuratete Testare": accuracy_score(etichete_test, pred_test),
        "Precizie": precision_score(etichete_test, pred_test),
        "Recall": recall_score(etichete_test, pred_test),
        "F1 Score": f1_score(etichete_test, pred_test),
        "AUC": scor_auc
    })

    # Salveaza predictii/probabilitati pentru grafice
    predictii[nume] = (pred_train, pred_test)
    probabilitati[nume] = prob_test
    matrici_confuzie[nume] = (etichete_test, pred_test)
```

Screenshot: loop-ul de train și evaluare.

6. Evaluarea performanței modelelor

Evaluarea modelelor a fost făcută prin următoarele metrici specifice clasificatoarelor

- **Acuratețe:** Procentul mailurilor clasificate corect, fie ele spam/not-spam.

- **Precision:** Mailuri clasificate ca spam care chiar sunt spam. Cât de „curate” sunt predicțiile pozitive.
- **Recall:** Proporţia de mailuri marcate ca spam care chiar sunt spam. Din totalul spamurilor, cate au fost identificate de model?
- **F1-score:** Media armonică a preciziei si recallului, măsoară balanța dintre cele două
- **AUC (area under ROC curve):** măsoară capacitatea modelului de a separa clasele la toate pragurile

Toate aceste metrici au fost calculate pe setul de test, mai jos se poate observa un tabel:

	Model	Acuratete Antrenare	Acuratete Testare	Precizie	Recall	F1 Score	AUC
0	Gradient Boosting	0.9689	0.9500	0.9488	0.9307	0.9396	0.9873
1	Random Forest	0.9665	0.9457	0.9614	0.9064	0.9331	0.9849
2	Regresie Logistica	0.9273	0.9232	0.9353	0.8769	0.9052	0.9735
3	Arbore de Decizie	0.9686	0.9109	0.9173	0.8648	0.8903	0.9215

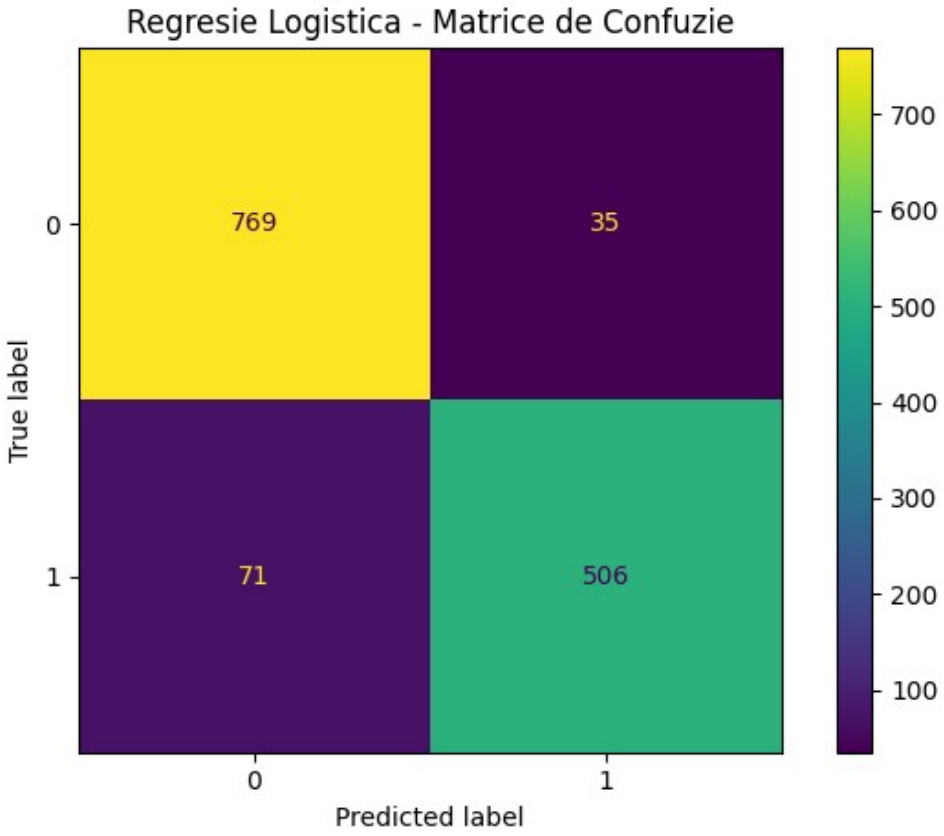
Screenshot: tabel performanță modele

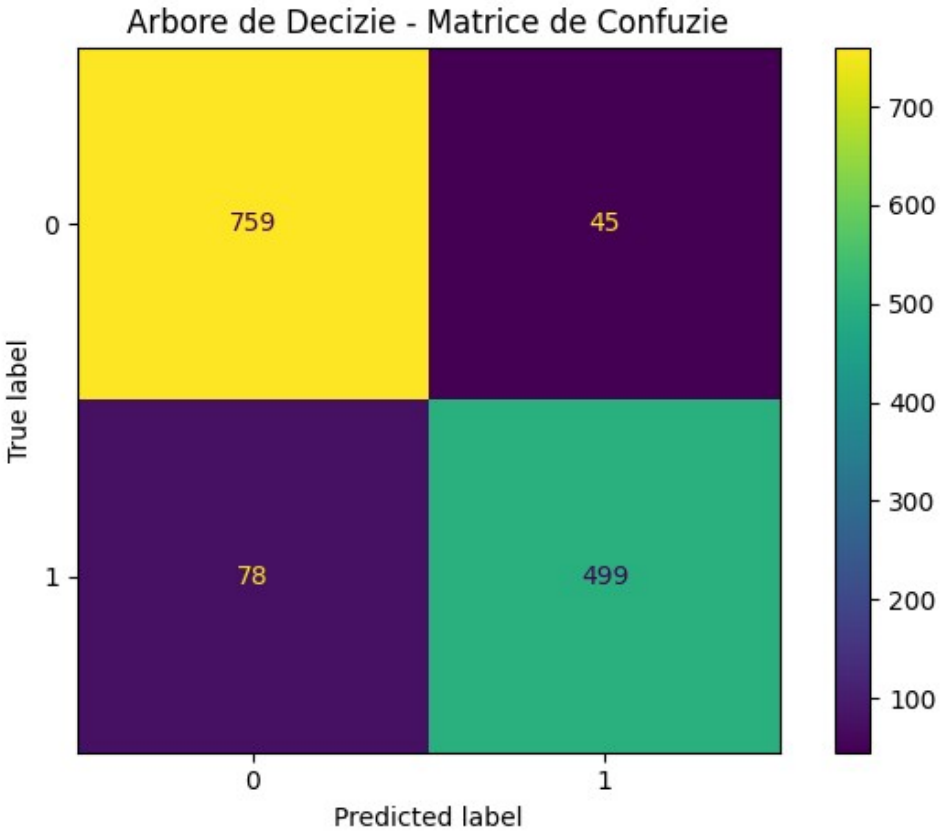
7. Vizualizarea Rezultatelor

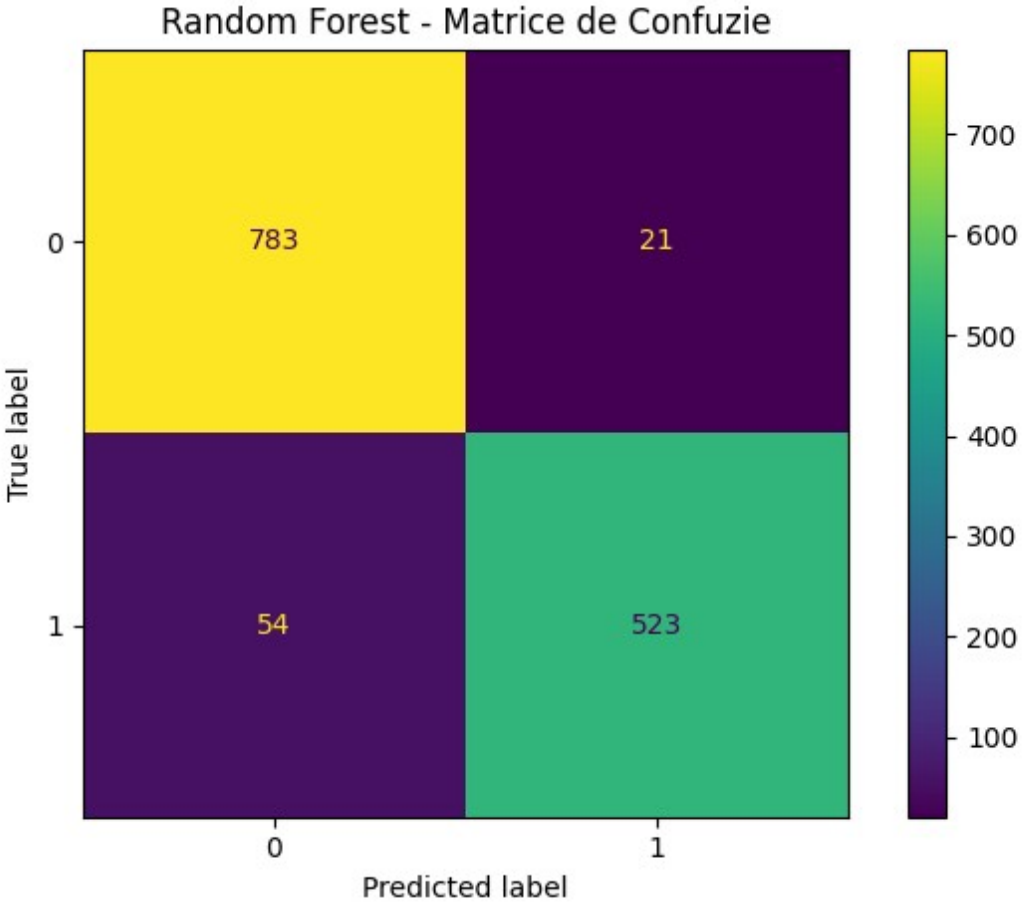
7.1 Matrice de Confuzie

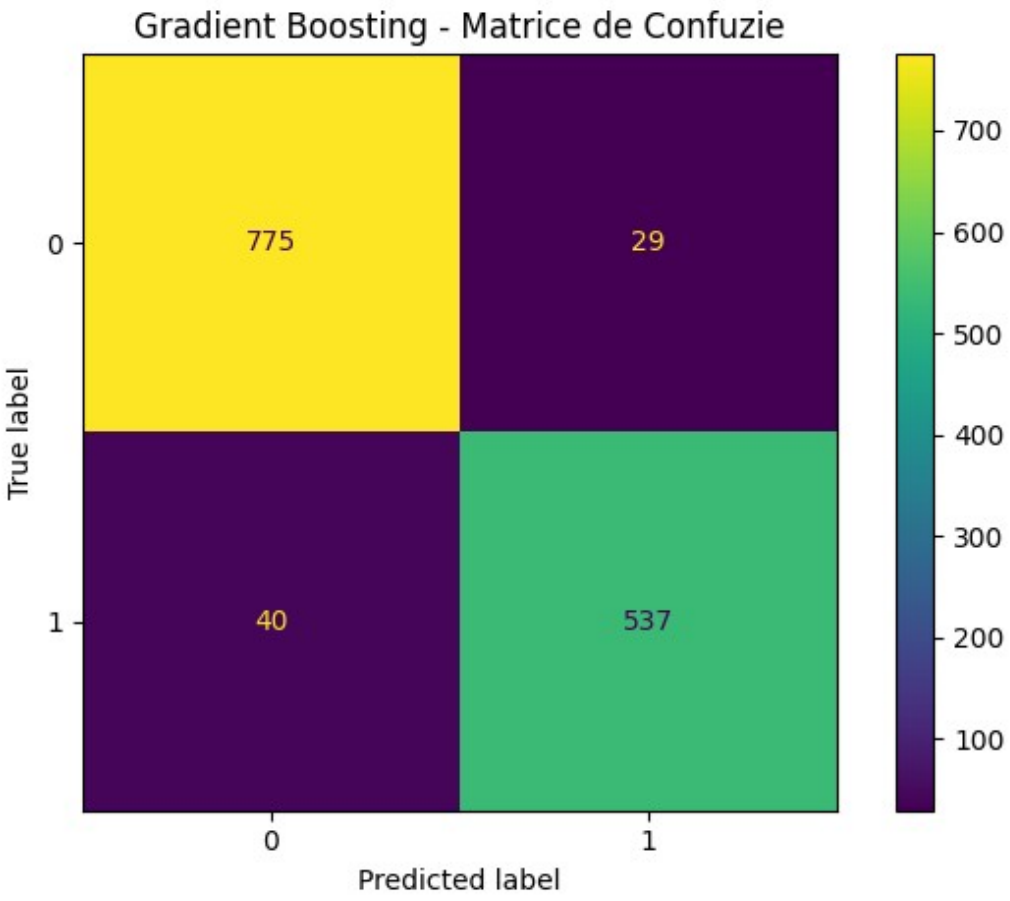
Matricea de confuzie e o bună metodă vizuală pentru a cuantifica numărul de predicții corecte, dar și tipurile de erori făcute:

Am generat matricea de confuzie pentru toate cele 4 modele:




Arbore de Decizie - Matrice de Confuzie
759
45
78
499
0
1
True label
0
1
Predicted label
700
600
500
400
300
200
100

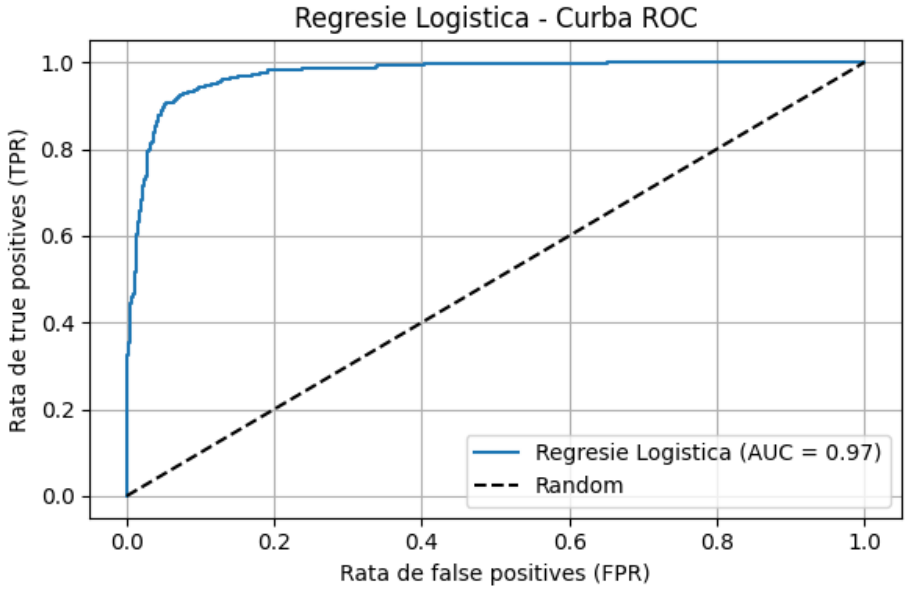

Random Forest - Matrice de Confuzie
783
21
54
523
0
1
True label
0
1
Predicted label
700
600
500
400
300
200
100

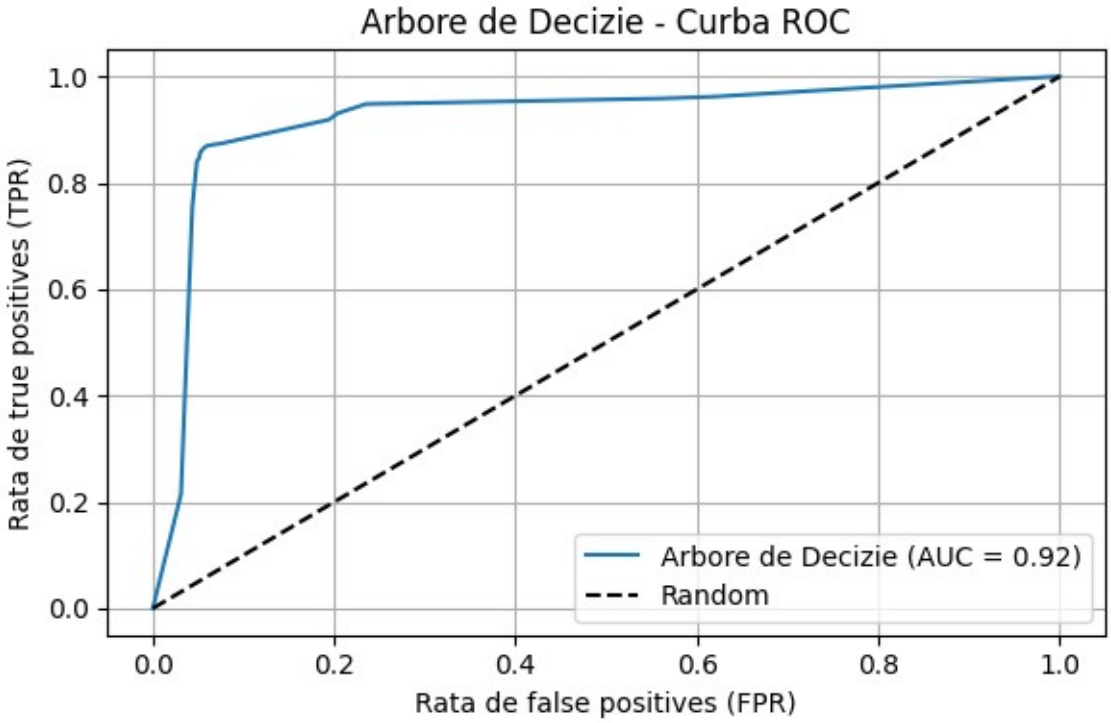


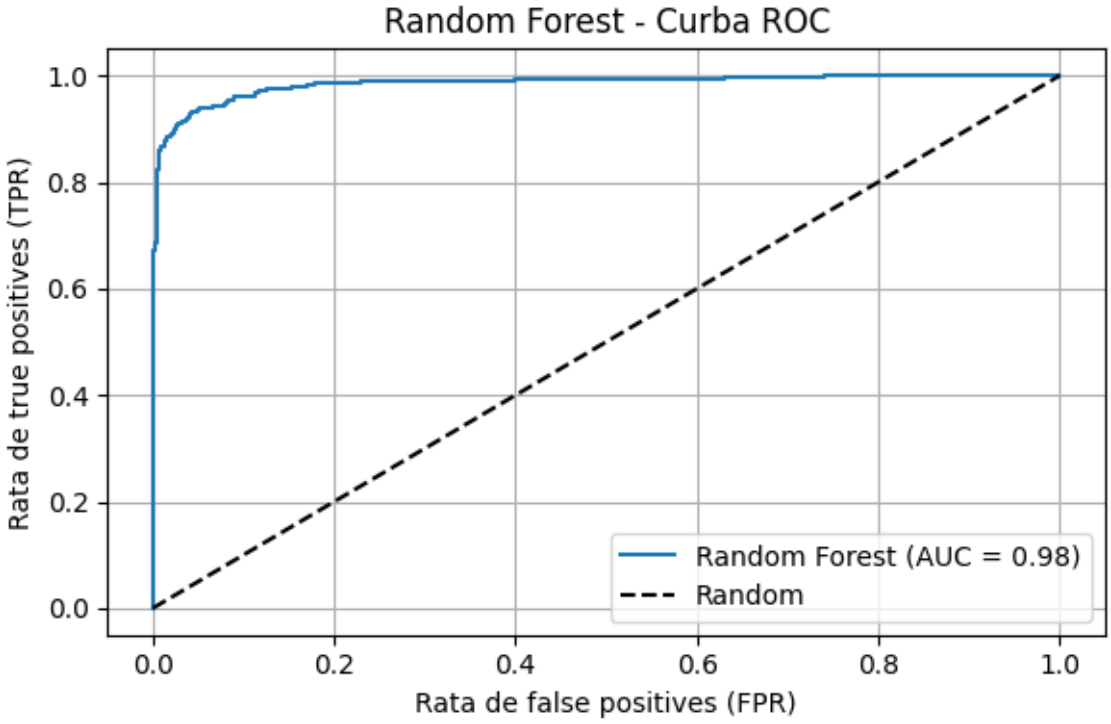
Se poate observa cum modelele bazate pe mai multi arbori au un numar mai mic de fals pozitive(non-spam marcate spam), comparat cu regresia și arborele de decizie. În acelaşi timp, fals negativele (spam marcat non-spam) sunt mai reduse la modelele ansamblu. Analizânnd metricile, noi am alege Gradient boosting, un spam pătruns in inbox este mult mai periculos decât un mail legitim ajuns in junk folder.

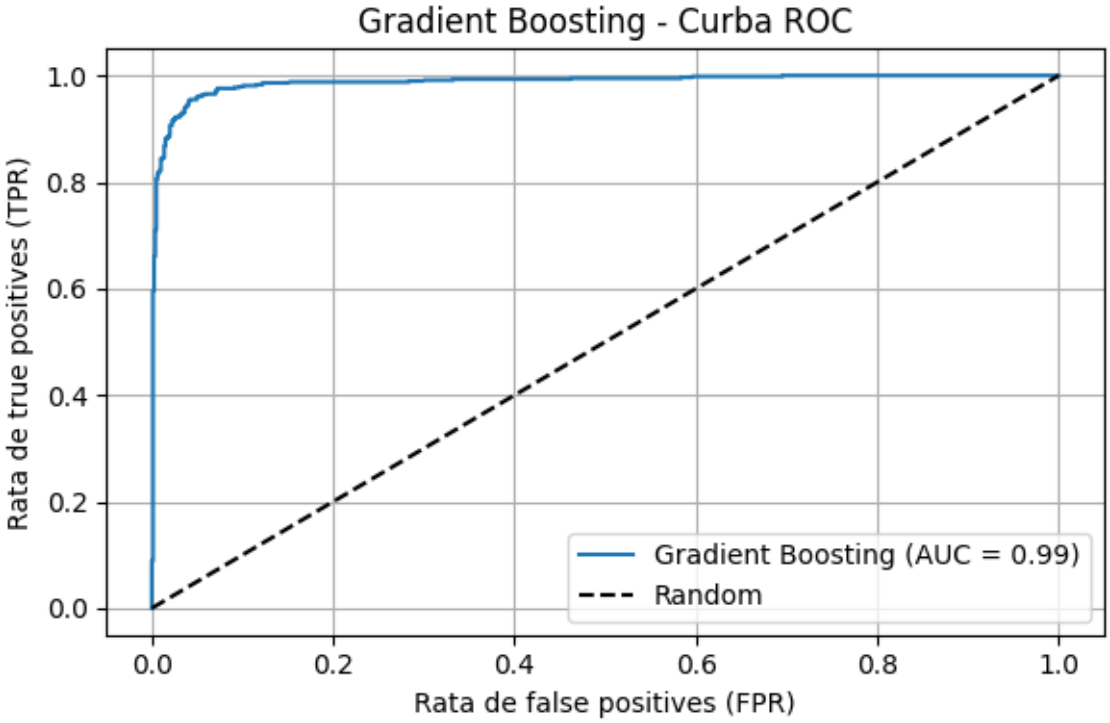
7.2 Curbe ROC individuale

Curbele ROC pentru fiecare model arată cât de bine distinge acesta între spam si non-spam, indiferent de pragul de decizie.
Cu cât curba e mai spre colț stânga sus, cu atat distincția dintre clase e mai bună:

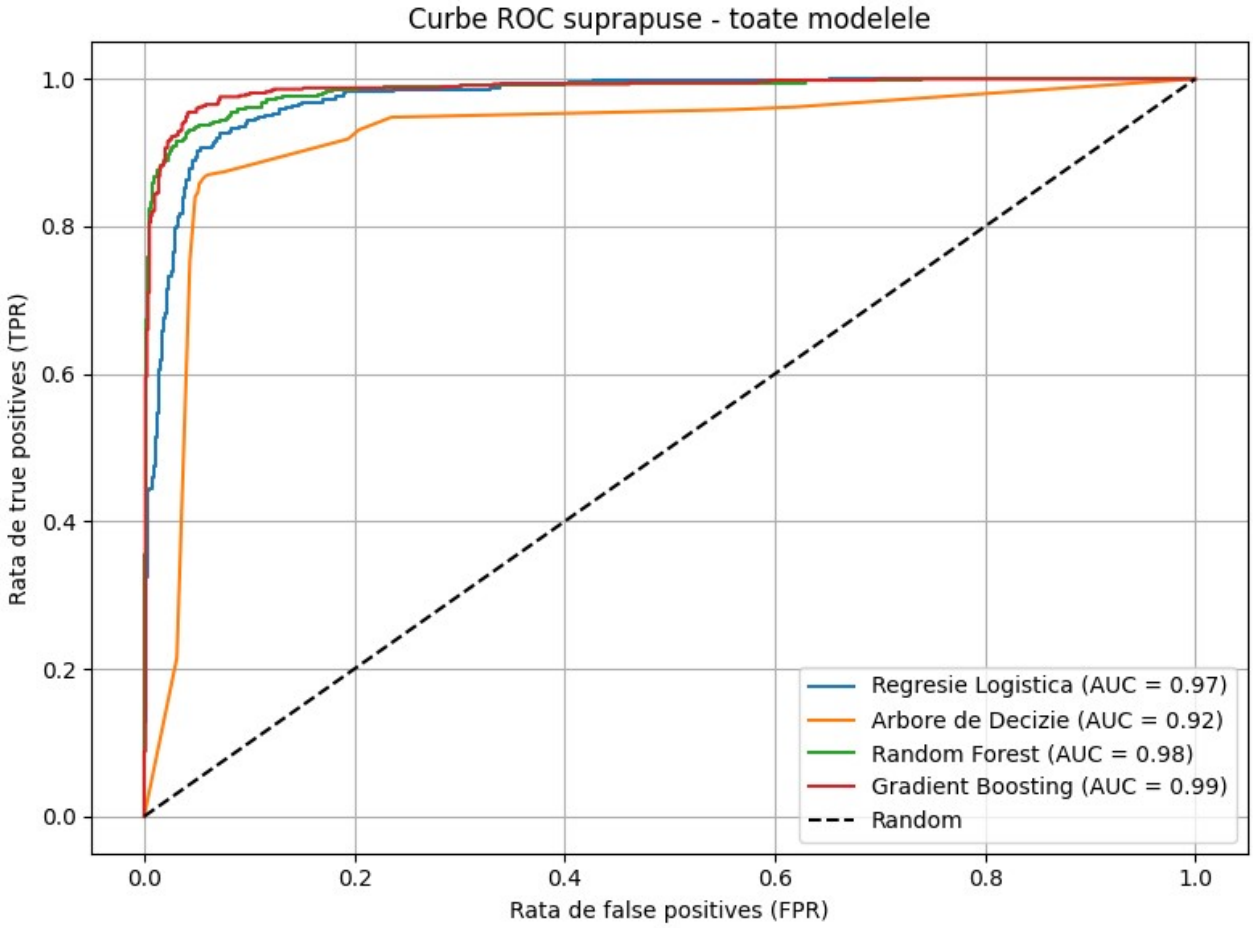



Arbore de Decizie - Curba ROC
Rata de true positives (TPR)
Rata de false positives (FPR)
Arbore de Decizie (AUC = 0.92)
Random
0.0
0.2
0.4
0.6
0.8
1.0


Random Forest - Curba ROC
Rata de true positives (TPR)
Rata de false positives (FPR)
Random Forest (AUC = 0.98)
Random
0.0
0.2
0.4
0.6
0.8
1.0

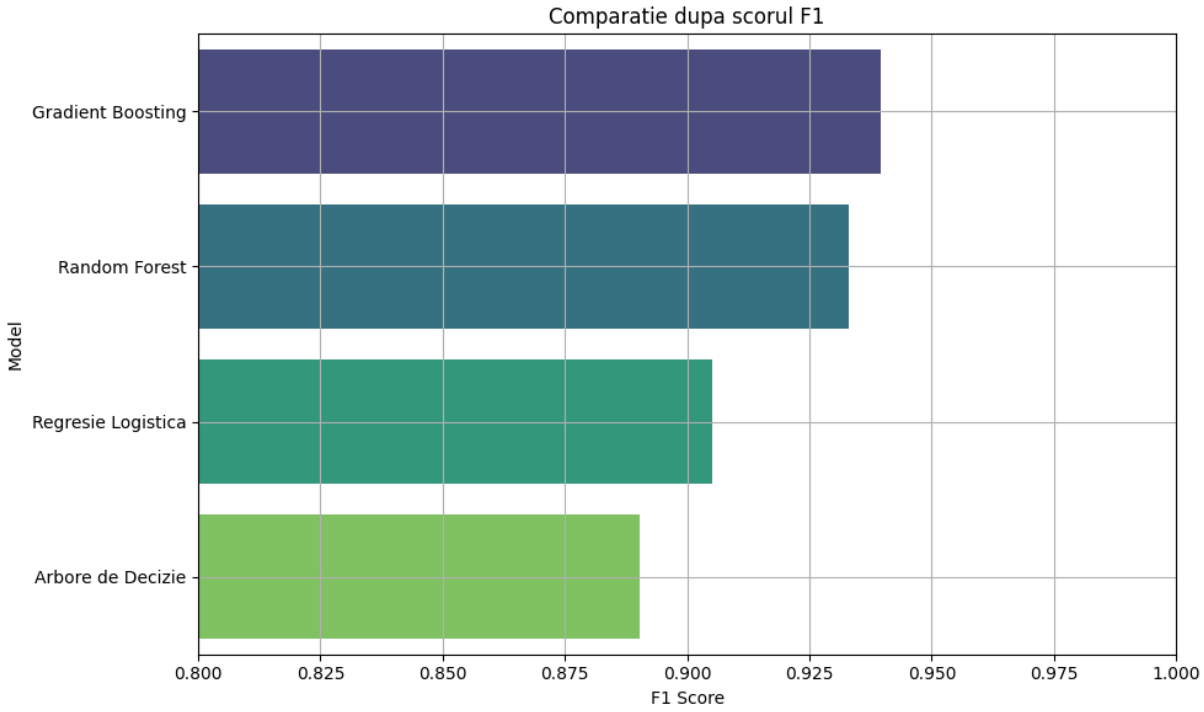


Toate curbele ROC suprapuse. Gradient Boosting are cea mai bună performanță, apoi Random Forest și Regresia Logistică. Arborele de Decizie rămâne sub restul.



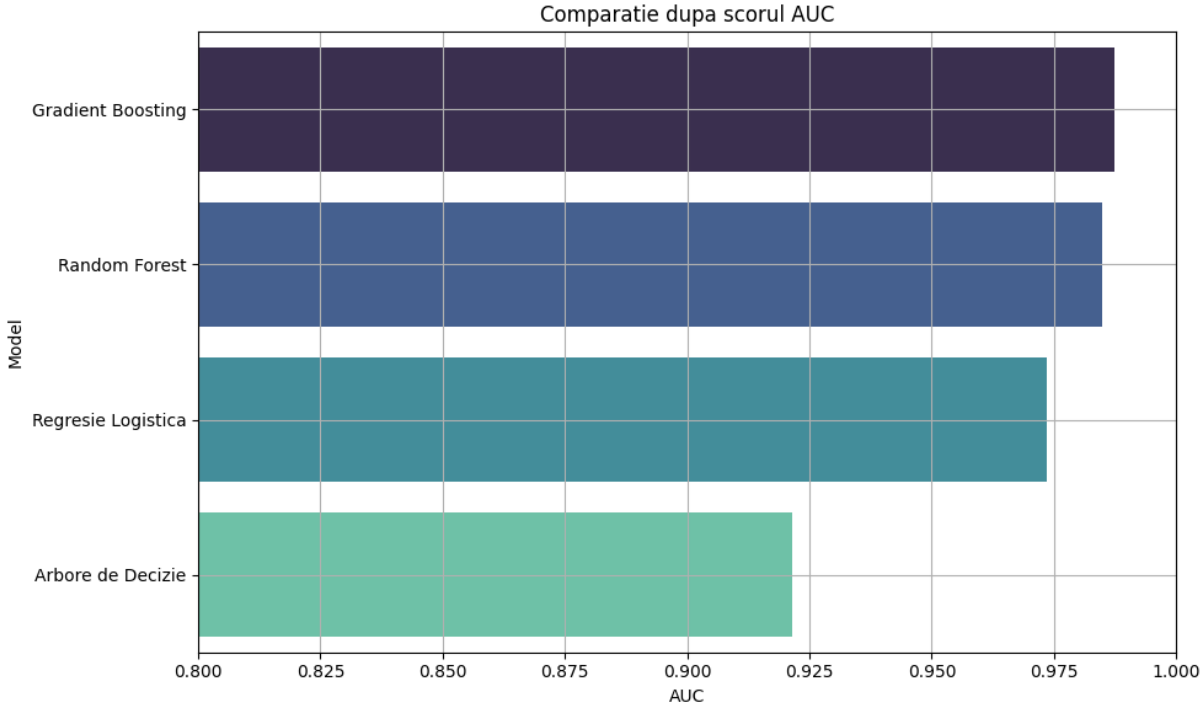
7.3 Comparaţie scor F1

Barplot cu scorurile F1 a celor 4 modele. Scor F1 ridicat arată un model care echilibrează bine precizia cu recall. Echilibru între etichetarea corectă a spamului şi evitarea de fals pozitive:



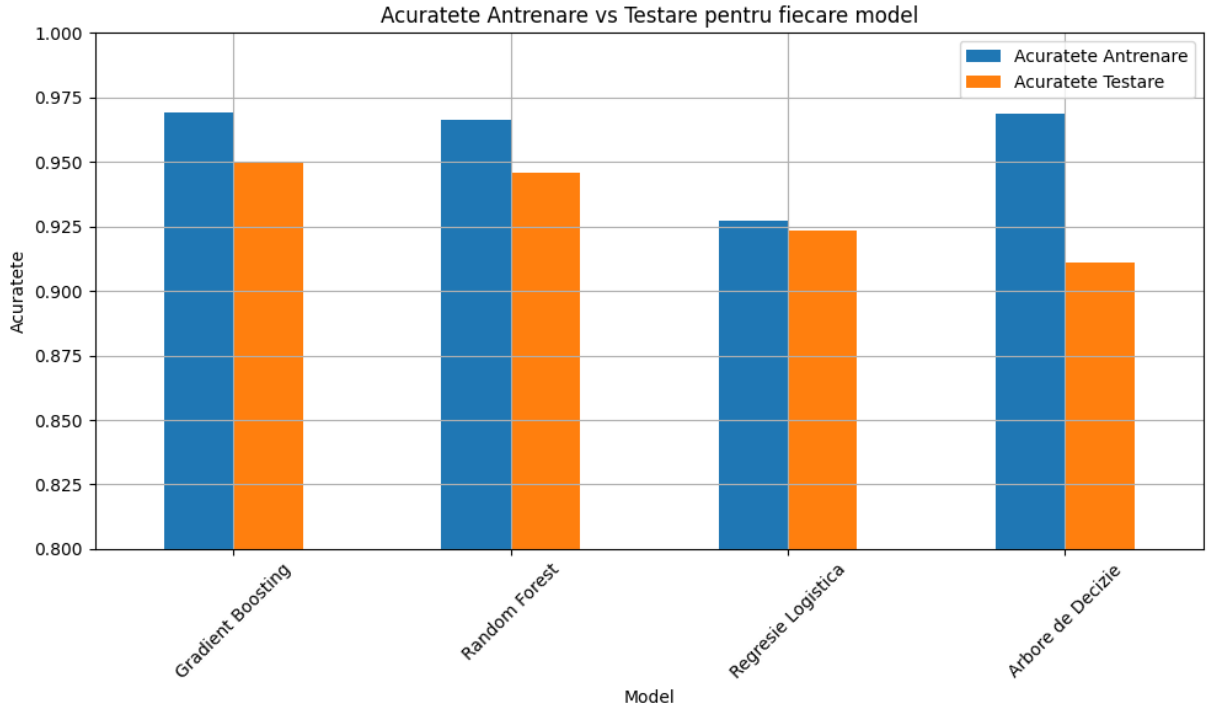
7.4 Comparaţie scor AUC

Barplot cu scorul AUC (Area Under the Curve). Scorul AUC reprezintă totalitatea curberi ROC, dar intr-o singură valoare. Un scor apropiat de 1 arată un model ce face bine diferența între clase, indiferent de pragul de decizie.



7.6 Overfitting Check (Acurateţe antrenare vs testare)

Scopul fiind testarea modelelor cu cât mai puţin fine-tuning, am inclus si un grafic cu perfomanţa la antrenare cât şi pe setul de test. Discrepanţă mare dintre acestea arată că modelul a invăţat mecanic setul de train şi nu are capacitate de generalizare pe test.



8. Testare practică pe un email

To provide a concrete example, a random email from the test set was selected, and its true label was revealed. Each trained model predicted the label and output the probability of spam. This demonstrates how models behave in a realistic, unseen scenario and can be used for hands-on evaluation.

Pentru a simula cum se comporta modelele într-un scenariu cât mai aproape de realitate, am introdus si o mică probă pe un e-mail aleator din setul de test.

Screenshot: cod probă

```
#
#6 Test final random pe un email din setul de test

i = np.random.randint(0, date_test.shape[0])
email_vector = date_test[i].reshape(1, -1)


print("\nTest random: predictii pentru un email aleator din setul de test")

eticheta_reala = etichete_test.iloc[i] if hasattr(etichete_test, "iloc") else list(etichete_test)[i
print(f"Eticheta reala: {'spam' if eticheta_reala == 1 else 'non-spam'}")

for nume, model in modele.items():
    pred = model.predict(email_vector)[0]
    prob = model.predict_proba(email_vector)[0][1]
    print(f"- {nume}: {'spam' if pred == 1 else 'non-spam'} (probabilitate spam: {prob:.2%})")
```

```
Test random: predictii pentru un email aleator din setul de test
Eticheta reala: spam
- Regresie Logistica: spam (probabilitate spam: 93.27%)
- Arbore de Decizie: spam (probabilitate spam: 100.00%)
- Random Forest: spam (probabilitate spam: 89.89%)
- Gradient Boosting: spam (probabilitate spam: 98.06%)
```

```
Test random: predictii pentru un email aleator din setul de test
Eticheta reala: spam
- Regresie Logistica: spam (probabilitate spam: 59.13%)
- Arbore de Decizie: spam (probabilitate spam: 100.00%)
- Random Forest: spam (probabilitate spam: 66.93%)
- Gradient Boosting: spam (probabilitate spam: 71.99%)
```

Screenshot: Output la 2 probe

9. Concluzie

Proiectul a demonstrat eficiența modelelor de ML in clasificarea e-mailurilor ca spam/non-spam. Am folosit doar atribute extrase din textul mesajelor. Gradient boosting si Random Forest s-au descurcat cel mai bine dupa scorul F1 si AUC, cu putinii parametri pe care i-am setat. Spambase a permis o analiză rapidă fară a mai fi nevoie de feature engineering.

11. Limitări și posibile îmbunătățiri.

Limitările principale ale proiectului sunt lipsa de fine-tuning a hyperparametrilor și utilizarea setului de date așa cum a venit, fără prea multe modificări. Vârsta setului de date probabil are și ea o influență, limbajul s-a mai schimbat putin de atunci.

Un pas următor ar fi să folosesc acelasi KDD la un set mult mai mare și mai complex, nu doar pe baza textului din body. Un set pe care-l am in minte este dump-ul de .eml de la Enron, aproximativ 88.000 de e-mailuri raw. As extrage și date din headere precum DKIM, DMARC, SPF, discrepanțe dintre „from” și „reply-to” (posibil spoofing) etc. Asta ar implica extragere și serializarea acestor atribute.

Intr-un final am ales Spambase fix pentru nivelul aproape nul de feature engineering necesar, scopul fiind pe antrenarea și evaluarea diferitor modele decât feature engineering.

11. Bibliografie

- UCI Machine Learning Repository: https://archive.ics.uci.edu/dataset/94/spambase
- documentație scikit-learn: https://scikit-learn.org/