Costea Carmen Andreea 321AA

Problema de optimizare neconstransa in invatare aprofundata

1. **Sarcina de invatare**: clasificare binara
   Obiectivul este de a invata un model care sa prezica dacă un pacient sufera sau nu de boli cardiovasculare, folosind caracteristici medicale extrase din baza de date.
2. **Detalii despre Baza de Date utilizata:**
   - Am utilizat Baza de Date **Heart Disease Dataset** de pe Kagle:https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset.
   - Fisierul heart.csv contine caracteristici pentru fiecare pacient:varsta, sexul, tipul durerii in piept, presiunea sangelui, nivelul de colesterol,glicemia, rezultate ECG, bataile inimii in repaus(nr maxim de batai pe minut), angina pectorală indusă de efort, coborarea segmentului ST la testul de efort, forma pantei segmentului ST si eticheta de clasificare(pacient cu sau fara boli la inima).
   - Am selectat aleatoriu 200 de exemple:160 pentru antrenare si 40 pentru testare.
   - Coloanele cu date categorice:Sex, ChestPainType, RestingECG, ExerciseAngina, ST_Slope au fost convertite in valori numerice.
3. **Algoritmii de optimizare implementati**
   Am implementat trei metode de optimizare:
   a. **Metoda Gradient:**
      - Toate exemplele sunt folosite la fiecare actualizare a greutatilor.
      - Direcția de optimizare este data de gradientul intregii functii de pierdere.
      - Functia de activare folosita la stratul ascuns este Soft++, avand un numar m=20 neuroni.
      - Functia de activare la ieșire este Sigmoid.
      - Functia de pierdere utilizata este Entropie Incrucisata Binara.
      - Calculul derivatei functiei de pierdere(in combinatie cu y dat de functia Sigmoid):**delta=y-e_train**, este realizata prin calcul mathematic, dupa cum urmeaza:

Costea Carmen Andreea 321AA

$$L(e,y) = -\frac{1}{N}\sum_{i=1}^{N}\left(e_i \log(y_i) + (1-e_i)\log(1-y_i)\right)$$

Dar $y_i = \sigma(z_i) = \frac{1}{1+e^{-z_i}}$ → am aplicat funcția sigmoid pe vectorul de predicții pt. a avea numere în intervalul $(0,1)$

$$\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_i}$$

dar
$$\left.\begin{aligned} \frac{\partial L}{\partial y_i} &= -\left(\frac{e_i}{y_i} - \frac{1-e_i}{1-y_i}\right) \\ \frac{\partial y_i}{\partial z_i} &= y_i(1-y_i) \end{aligned}\right\} \Rightarrow$$

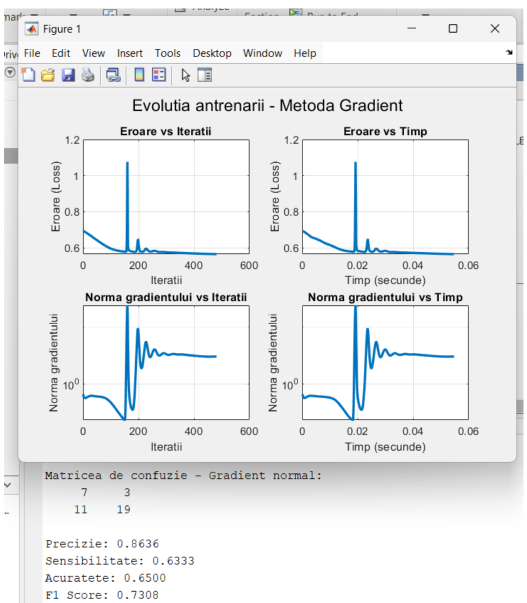
$$\frac{\partial L}{\partial z_i} = \left(\frac{1-e_i}{1-y_i} - \frac{e_i}{y_i}\right)\cdot y_i(1-y_i)$$

$$= y_i(1-e_i) - e_i(1-y_i)$$

$$= y_i - \cancel{y_i e_i} - e_i + \cancel{e_i y_i}$$

$$\Rightarrow \boxed{\frac{\partial L}{\partial z_i} = y_i - e_i}$$

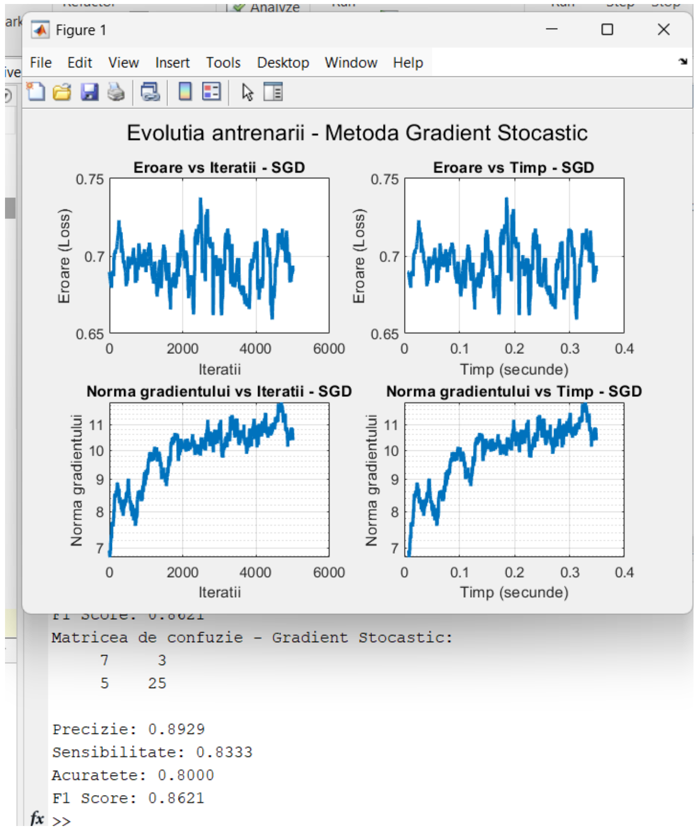
⇒ delta = y − e_train − derivata funcției de pierdere combinată cu funcția sigmoid!



In cadrul Metodei Gradient, s-a observat o scadere initiala a functiei obiectiv, urmată de o crestere brusca a erorii si a normei gradientului. Acest fenomen se datoreaza unui pas de invatare (alfa) fix, care, combinat cu un gradient mare local, a produs un salt excesiv ("overshoot") pe direcția optimizarii. Ulterior, reteaua neuronală a reusit să corecteze saltul, funtia obiectiv si norma gradientului stabilizandu-se treptat.

b. Metoda Gradient Stocastic

- Actualizarea greutatilor se face folosind cate un singur exemplu la fiecare iterație.
- Această metodă introduce variație mai mare in gradient si poate evita minime locale.
- Aceeasi arhitectura de retea ca la metoda Gradientului.

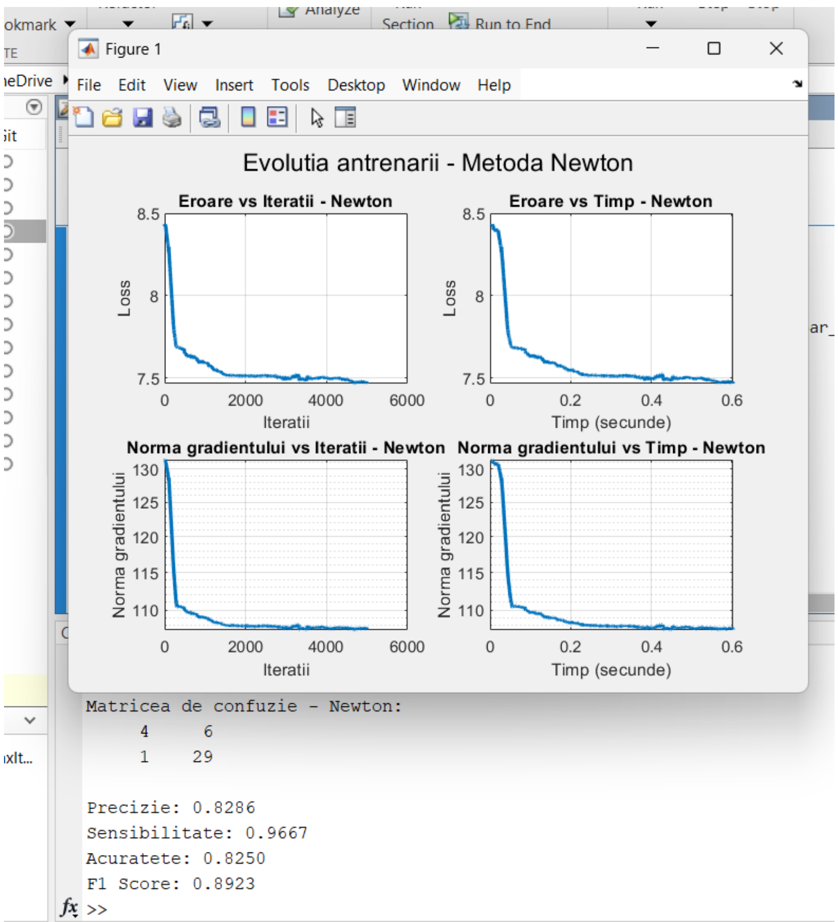


Aplicand o netezire (moving average) asupra erorii și normei gradientului in metoda Gradient Stocastic, se observa o tendinta generala de stabilizare a functiei obiectiv în jurul valorii 0.7. Norma gradientului creste usor in primele faze ale antrenarii, dar ulterior se stabilizeaza, indicand o convergenta partiala caracteristica optimizarii stocastice. Fluctuatiile raman prezente datorita naturii aleatoare a metodei SGD.

c. Metoda Newton

- Se folosește gradientul și Hessiana functiei de pierdere pentru un pas mai precis de optimizare.
- Pentru a evita probleme de instabilitate numerica (Hessiana aproape singulara), s-a aplicat regularizarea: $H=H+\lambda I$, cu $\lambda=10^{-4}$.
- Metoda Newton converge in mai putine iteratii, dar costul per iteratie este mai mare.

- In cadrul metodei Newton nu s-a utilizat activarea Soft++ deoarece optimizarea s-a realizat direct pe modelul logistic simplu, fara strat ascuns, cu functia sigmoid aplicata asupra produsului scalar dintre vectorul de greutati si caracteristicile exemplului.



Evoluția rapida și stabilizarea timpurie sunt caracteristice metodei Newton datorita folosirii informatiei de ordinul doi (Hessiana), ceea ce permite realizarea unor pasi de optimizare foarte eficienti spre minimul functiei obiectiv. Totusi, costul computational al fiecarei iteratii Newton este mai ridicat decat in metodele Gradient si Gradient Stocastic, ceea ce poate limita aplicabilitatea acestei metode în probleme de dimensiuni foarte mari.

- Pentru cele 3 metode am verificat:
  <u>Matricea de confuzie :</u>compara predictiile modelului fata de etichetele reale(clasificare binara:0 fara boala, 1 cu boala).
  <u>Indicatori de performanta:</u>
  -PRECIZIA:cat de des are dreptate modelul(mare=>modelul face putine alarme false-putine personae sanatoase sunt declarate bolnave)

-SENSIBILITATEA:din pacientii bolnavi, pe cati i-a detectat correct modelul(mare=>modelul detecteaza bolile bine)
-ACURATETEA:cate predictii corecte face modelul in total(mare=>modelul face multe predictii corecte, indifferent de clasa-0 sau 1)
-SCORUL F:compromis intre precizie si sensibilitate(mare=> modelul are echilibru bun intre a detecta corect boala si a nu da alarme false)

- Pentru fiecare metodă s-au generat grafice:
  -Eroarea (loss) în functie de numarul de iteratii
  -Eroarea (loss) în functie de timp
  -Norma gradientului în functie de numarul de iteratii
  -Norma gradientului în functie de timp

4. **Concluzii**

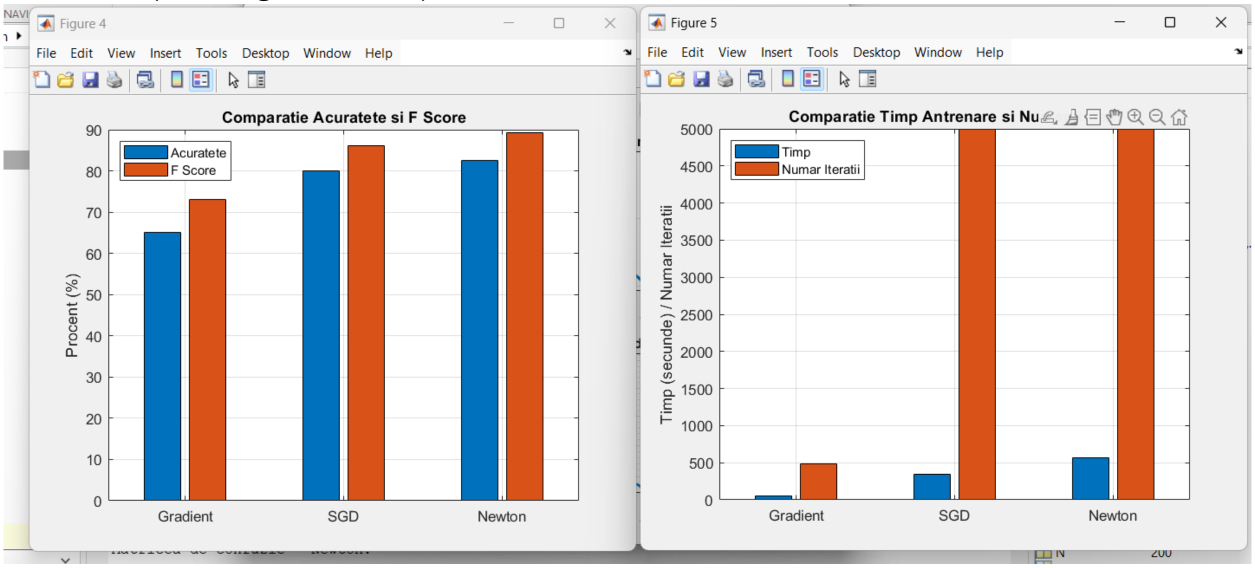
In cadrul metodelor Gradient si Gradient Stocastic, optimizarea s-a realizat asupra unei retele neuronale simple, cu un strat ascuns ce utilizeaza functia de activare Soft++ si functia sigmoid pentru iesirea finala. Aceasta structura a permis modelului sa invete relatii nelineare complexe intre caracteristici.

In schimb, metoda Newton a fost aplicata direct pe regresia logistica standard, fara strat ascuns, folosind doar functia sigmoid aplicata asupra produsului scalar dintre vectorul de greutati si datele de intrare. Astfel, metoda Newton optimizeaza un model simplu, liniar in caracteristici, ceea ce ii permite o convergenta mai rapida, dar cu o capacitate redusa de modelare a relatiilor nelineare.

Din diagramele de mai jos, se observa ca:

- Metoda Gradient Stocastic obtine cele mai bune performante in ceea ce priveste atat acuratetea, cat si F Score-ul, demonstrand un echilibru eficient intre precizie si sensibilitate. Gradientul Stocastic necesita cele mai multe iteratii, dar fiecare pas este rapid, ceea ce mentine un timp de antrenare scazut.
- Gradientul normal are performante usor mai slabe, fiind mai sensibil la alegerea hiperparametrilor. Totusi, aceasta metoda prezinta un compromis echilibrat intre numarul de iteratii si timpul total de antrenare.
- Metoda Newton converge rapid , insa timpul total de antrenare nu este intotdeauna cel mai mic din cauza costului ridicat al calculului Hessienei.
- Metoda Gradient normal s-a oprit dupa aproximativ 500 de iteratii, deoarece criteriul de oprire pe baza diferentei functiei obiectiv a fost atins rapid.
  Gradientul Stocastic si metoda Newton au atins numarul maxim de 5000 de iteratii, din cauza fluctuatiilor specifice SGD si a criteriului mai strict de convergenta folosit

la Newton (norma gradientului).



- Asadar, alegerea metodei optime depinde de compromisurile acceptate intre viteza de convergenta, complexitatea computationala si performanta obtinuta in clasificare, fiecare metoda prezentand avantaje si limitari specifice in functie de contextul aplicatiei.

Anexa cod matlab

ALEGERE BAZA DE DATE SI IMPARTIRE DATE(ANTRENARE+TESTARE)

```
T = readtable('heart.csv');

% Transformare coloane categorice(stringuri) in coloane numerice
T.Sex = double(categorical(T.Sex));%F=1,M=2
T.ChestPainType = double(categorical(T.ChestPainType));%ASY=1,ATA=2,NAP=3
T.RestingECG = double(categorical(T.RestingECG));%Normal=2,ST=3
T.ExerciseAngina = double(categorical(T.ExerciseAngina));%N=1,Y=2
T.ST_Slope = double(categorical(T.ST_Slope));%Flat=2, Up=3
%disp(T)
A = T{:,1:end-1}; % input
e = T{:,end};     % eticheta (HeartDisease)

[N_total, n] = size(A);%dimensiune originala
idx = randperm(N_total, 200); % aleg aleatoriu 200 de exemple
A = A(idx, :); % pastrez doar 200 de exemple
e = e(idx, :);

N = 200; % nr de exemple cu care lucrez
```

```
N_train = 160;% 80% pt antrenare
N_test = 40;% 20% pt testare

% Seturi de antrenare
A_train = A(1:N_train, :);
e_train = e(1:N_train, :);
%Seturi de testare
A_test = A(N_train+1:end, :);
e_test = e(N_train+1:end, :);

% Ā = [A, 1]
Abar_train = [A_train, ones(N_train,1)];
Abar_test = [A_test, ones(N_test,1)];
```

Retea Neuronala

```
m=20;%nr neuroni strat ascuns
a=1;
b=2;%parametrii pt Soft++

[N_train, n_plus_1] = size(Abar_train); % n+1 caracteristici
n = n_plus_1 - 1; % caracteristici reale

rng(0); % pentru reproductibilitate
X = randn(n_plus_1, m) * 0.01; % greutati strat ascuns (Abar -> strat ascuns)
x = randn(m,1) * 0.01;         % greutati strat ascuns -> iesire

Z=Abar_train*X;% ĀX
H=softplusplus(Z,a,b);% g(ĀX) - activare strat ascuns
y1=H*x;%vector de predictii
y=sigmoid(y1);%pt clasificare binara vreau ca y sa aiba valori intre 0 si 1
loss=loss_crossentropy(e_train,y);
```

METODA GRADIENT

Setari

```
alfa = 0.003;
epsilon = 1e-5;
maxIter = 5000;
m = 20;
a = 1;
b = 2;

% Antrenare cu metoda gradientului
```

```
    [X_final, x_final, errors, norms, times] = gradient(Abar_train, e_train, m, a, b, alfa, epsilon, maxIter);
    figure;
    % 1. Eroare vs Iteratii
    subplot(2,2,1);
    plot(1:length(errors), errors, 'LineWidth', 2);
    xlabel('Iteratii');
    ylabel('Eroare (Loss)');
    title('Eroare vs Iteratii');
    grid on;
    % 2. Eroare vs Timp
    subplot(2,2,2);
    plot(times, errors, 'LineWidth', 2);
    xlabel('Timp (secunde)');
    ylabel('Eroare (Loss)');
    title('Eroare vs Timp');
    grid on;
    % 3. Norma gradientului vs Iteratii
    subplot(2,2,3);
    semilogy(1:length(norms), norms, 'LineWidth', 2);
    xlabel('Iteratii');
    ylabel('Norma gradientului');
    title('Norma gradientului vs Iteratii');
    grid on;
    % 4. Norma gradientului vs Timp
    subplot(2,2,4);
    semilogy(times, norms, 'LineWidth', 2);
    xlabel('Timp (secunde)');
    ylabel('Norma gradientului');
    title('Norma gradientului vs Timp');
    grid on;
    sgtitle('Evolutia antrenarii - Metoda Gradient');

    %Performanta sarcinii de invatare
    [precizie_grad, sensibilitate_grad, acuratete_grad, F1_grad, C_grad] = evaluare_model(Abar_test, e_test, X_final, x_final, a, b);

    disp('Matricea de confuzie - Gradient normal:');
    disp(C_grad);
    fprintf('Precizie: %.4f\n', precizie_grad);
    fprintf('Sensibilitate: %.4f\n', sensibilitate_grad);
    fprintf('Acuratete: %.4f\n', acuratete_grad);
    fprintf('F1 Score: %.4f\n', F1_grad);
```

METODA GRADIENT STOCASTICA

Costea Carmen Andreea 321AA

Setari

```
    alfa = 0.003;
    epsilon = 1e-5;
    maxIter = 5000;
    m = 20;
    a = 1;
    b = 2;
    % Antrenare cu metoda gradientului stocastic
    [X_sgd, x_sgd, errors_sgd, norms_sgd, times_sgd] =
gradient_stocastic(Abar_train, e_train, m, a, b, alfa, epsilon, maxIter);
    window = 200; % fereastra de mediere-vad mai bine grafic
    errors_sgd = movmean(errors_sgd, window);
    norms_sgd = movmean(norms_sgd, window);
    figure;
    % 1. Eroare vs Iteratii
    subplot(2,2,1);
    plot(1:length(errors_sgd), errors_sgd, 'LineWidth', 2);
    xlabel('Iteratii');
    ylabel('Eroare (Loss)');
    title('Eroare vs Iteratii - SGD');
    grid on;
    % 2. Eroare vs Timp
    subplot(2,2,2);
    plot(times_sgd, errors_sgd, 'LineWidth', 2);
    xlabel('Timp (secunde)');
    ylabel('Eroare (Loss)');
    title('Eroare vs Timp - SGD');
    grid on;
    % 3. Norma gradientului vs Iteratii
    subplot(2,2,3);
    semilogy(1:length(norms_sgd), norms_sgd, 'LineWidth', 2);
    xlabel('Iteratii');
    ylabel('Norma gradientului');
    title('Norma gradientului vs Iteratii - SGD');
    grid on;
    % 4. Norma gradientului vs Timp
    subplot(2,2,4);
    semilogy(times_sgd, norms_sgd, 'LineWidth', 2);
    xlabel('Timp (secunde)');
    ylabel('Norma gradientului');
    title('Norma gradientului vs Timp - SGD');
    grid on;
    sgtitle('Evolutia antrenarii - Metoda Gradient Stocastic');

    %Performanta sarcinii de invatare
    [precizie_sgd, sensibilitate_sgd, acuratete_sgd, F1_sgd, C_sgd] =
evaluare_model(Abar_test, e_test, X_sgd, x_sgd, a, b);
```

```
    disp('Matricea de confuzie - Gradient Stocastic:');
    disp(C_sgd);
    fprintf('Precizie: %.4f\n', precizie_sgd);
    fprintf('Sensibilitate: %.4f\n', sensibilitate_sgd);
    fprintf('Acuratete: %.4f\n', acuratete_sgd);
    fprintf('F1 Score: %.4f\n', F1_sgd);
```

METODA NEWTON

```
    epsilon = 1e-5;
    maxIter = 5000;
    % Antrenare Newton
    [w_newton, errors_newton, norms_newton, times_newton] = Newton(Abar_train',
e_train, epsilon, maxIter);
    % Ploturi evolutie Newton
    window = 200; % fereastra de mediere-vad mai bine grafic
    errors_newton = movmean(errors_newton, window);
    norms_newton = movmean(norms_newton, window);
    figure;
    % 1. Eroare vs Iteratii
    subplot(2,2,1);
    plot(1:length(errors_newton), errors_newton, 'LineWidth', 2);
    xlabel('Iteratii');
    ylabel('Loss');
    title('Eroare vs Iteratii - Newton');
    grid on;
    % 2. Eroare vs Timp
    subplot(2,2,2);
    plot(times_newton, errors_newton, 'LineWidth', 2);
    xlabel('Timp (secunde)');
    ylabel('Loss');
    title('Eroare vs Timp - Newton');
    grid on;
    % 3. Norma gradientului vs Iteratii
    subplot(2,2,3);
    semilogy(1:length(norms_newton), norms_newton, 'LineWidth', 2);
    xlabel('Iteratii');
    ylabel('Norma gradientului');
    title('Norma gradientului vs Iteratii - Newton');
    grid on;
    % 4. Norma gradientului vs Timp
    subplot(2,2,4);
    semilogy(times_newton, norms_newton, 'LineWidth', 2);
    xlabel('Timp (secunde)');
    ylabel('Norma gradientului');
    title('Norma gradientului vs Timp - Newton');
```

```
    grid on;
    sgtitle('Evolutia antrenarii - Metoda Newton');

    % Predictii pe setul de testare - metoda Newton
    m_test = size(Abar_test,1);
    functii_test = zeros(m_test,1);
    for i = 1:m_test
        functii_test(i) = sigmoid(w_newton' * Abar_test(i,:)');
    end

    predictii_test_newton = double(functii_test >= 0.5);

    % Matricea de confuzie si performanta - Newton
    C_newton = confusionmat(e_test, predictii_test_newton);

    RP = C_newton(2,2);
    FP = C_newton(1,2);
    FN = C_newton(2,1);
    RN = C_newton(1,1);

    precizie_newton = RP / (RP + FP);
    sensibilitate_newton = RP / (RP + FN);
    acuratete_newton = (RP + RN) / (RP + RN + FP + FN);
    F1_newton = 2 * (precizie_newton * sensibilitate_newton) / (precizie_newton +
sensibilitate_newton);
    % Afisare rezultate
    disp('Matricea de confuzie - Newton:');
    disp(C_newton);
    fprintf('Precizie: %.4f\n', precizie_newton);
    fprintf('Sensibilitate: %.4f\n', sensibilitate_newton);
    fprintf('Acuratete: %.4f\n', acuratete_newton);
    fprintf('F1 Score: %.4f\n', F1_newton);
```

Comparare Metode

Comparatie Performante- Acuratete si F Score

```
    metode = {'Gradient', 'SGD', 'Newton'};
    acuratete = [acuratete_grad, acuratete_sgd, acuratete_newton] * 100;
    F1_scores = [F1_grad, F1_sgd, F1_newton] * 100;

    figure;
    bar([acuratete; F1_scores]');
    ylabel('Procent (%)');
    title('Comparatie Acuratete si F Score');
    legend('Acuratete', 'F Score', 'Location', 'northwest');
    set(gca, 'XTickLabel', metode);
```

```
grid on;

% Comparatie metode - Timp si Numar Iteratii
timpuri = [times(end), times_sgd(end), times_newton(end)];
timpuri=timpuri*1000;% scalez pt a vedea grafic
numar_iteratii = [length(errors), length(errors_sgd), length(errors_newton)];
figure;
bar([timpuri; numar_iteratii]');
ylabel('Timp (secunde) / Numar Iteratii');
title('Comparatie Timp Antrenare si Numar Iteratii');
legend('Timp', 'Numar Iteratii', 'Location', 'northwest');
set(gca, 'XTickLabel', metode);
grid on;
```

Functii:

```
METODA GRADIENT
function [X, x, errors, norms, times] = gradient(Abar_train, e_train, m, a, b,
alfa, epsilon, maxIter)
    [N_train, n_plus_1] = size(Abar_train);
    rng(0); % pentru reproductibilitate
    X = randn(n_plus_1, m) * 0.01;
    x = randn(m, 1) * 0.01;

    errors = zeros(1, maxIter);
    norms = zeros(1, maxIter);
    times = zeros(1, maxIter);

    iter = 0;
    f0 = 1;
    f1 = 0;
    norma_gradient = inf;

    tic;

    while abs(f1 - f0) >= epsilon && iter < maxIter
        iter = iter + 1;
        f0 = f1;
        Z = Abar_train * X;
        H = softplusplus(Z, a, b);
        y1 = H * x;
        y = sigmoid(y1);
        f1 = loss_crossentropy(e_train, y);
        delta = y - e_train;%derivata fct. de pierdere combinata cu sigmoid
        % Gradient fata de x
        grad_x = H' * delta / N_train;
        % Gradient fata de X
        dH = (delta * x') / N_train;
        dSoft = d_softplusplus(Z, a, b);
        grad_X = Abar_train' * (dH .* dSoft);
```

```
        norma_gradient = sqrt(sum(grad_x.^2) + sum(grad_X(:).^2));%norma
gradientului

        % Metoda Gradient
        x = x - alfa * grad_x;
        X = X - alfa * grad_X;

        errors(iter) = f1;
        norms(iter) = norma_gradient;
        times(iter) = toc; % timpul curent la fiecare iteratie
    end
    % Trunchiere vectorii la numarul real de iteratii
    errors = errors(1:iter);
    norms = norms(1:iter);
    times = times(1:iter);

end
```

```
% Output g este o valoare intre 0 si 1
function g = sigmoid(z)
    g = 1.0 ./ (1.0 + exp(-z));
end
```

```
function L=loss_crossentropy(e,y)
N=length(e);
eps=1e-8;%eroare pt log(0)
suma=0;
for i=1:N
    suma=suma+(e(i)*log(y(i)+eps)+(1-e(i))*log(1-y(i)+eps));
end
L=(-1/N)*suma;
end
```

```
function g=softplusplus(z,a,b)
g=log(1+exp(a.*z))+(z./b)-log(2);
end
```

```
METODA GRADIENT STOCASTIC
function [X, x, errors, norms, times] = gradient_stocastic(Abar_train, e_train,
m, a, b, alfa, epsilon, maxIter)
    [N_train, n_plus_1] = size(Abar_train);
    rng(0); % reproducibilitate
    X = randn(n_plus_1, m) * 0.01;
    x = randn(m, 1) * 0.01;

    errors = zeros(1, maxIter);
    norms = zeros(1, maxIter);
    times = zeros(1, maxIter);
```

```
    iter = 0;
    f0 = 1;
    f1 = 0;
    norma_gradient = inf;

    tic;

    while abs(f1 - f0) >= epsilon && iter < maxIter
        iter = iter + 1;
        f0 = f1;
        % Aleg un exemplu aleatoriu
        i = randi(N_train); % indice aleator intre 1 si N_train
        Abar_i = Abar_train(i, :); % exemplul selectat
        e_i = e_train(i); % eticheta asociata
        Z_i = Abar_i * X;
        H_i = softplusplus(Z_i, a, b);
        y1_i = H_i * x;
        y_i = sigmoid(y1_i);
        % Loss doar pentru exemplul i (optional, pentru verificare)
        % f1_i = loss_crossentropy(e_i, y_i);
        delta_i = y_i - e_i; % derivata combinata pierdere+sigmoid

        % Gradient fata de x si X
        grad_x = H_i' * delta_i;
        dH_i = delta_i * x';
        dSoft_i = d_softplusplus(Z_i, a, b);
        grad_X = Abar_i' * (dH_i .* dSoft_i);
        norma_gradient = sqrt(sum(grad_x.^2) + sum(grad_X(:).^2));

        % Metoda Gradient
        x = x - alfa * grad_x;
        X = X - alfa * grad_X;

        % Evaluez loss pe toate datele (ca sa urmaresc evolutia corecta)
        Z_full = Abar_train * X;
        H_full = softplusplus(Z_full, a, b);
        y1_full = H_full * x;
        y_full = sigmoid(y1_full);
        f1 = loss_crossentropy(e_train, y_full);

        errors(iter) = f1;
        norms(iter) = norma_gradient;
        times(iter) = toc;
    end
    % Trunchiere vectorii la numarul real de iteratii
    errors = errors(1:iter);
    norms = norms(1:iter);
    times = times(1:iter);
end
```

```
function [precizia, sensibilitatea, acuratetea, F1_score, C] =
evaluare_model(Abar_test, e_test, X, x, a, b)
    % Forward propagation pe setul de testare
```

Costea Carmen Andreea 321AA

```
    Z_test = Abar_test * X;
    H_test = softplusplus(Z_test, a, b);
    y1_test = H_test * x;
    y_test = sigmoid(y1_test);
    predictii_test = double(y_test >= 0.5);% Predictii finale
    C = confusionmat(e_test, predictii_test);   % Matricea de confuzie

    RP = C(2,2);
    FP = C(1,2);
    FN = C(2,1);
    RN = C(1,1);

    precizia = RP / (RP + FP);
    sensibilitatea = RP / (RP + FN);
    acuratetea = (RP + RN) / (RP + RN + FP + FN);
    F1_score = 2 * (precizia * sensibilitatea) / (precizia + sensibilitatea);

end
```

```
function g_prime = d_softplusplus(z, a, b)
    g_prime = (a * exp(a*z)) ./ (1 + exp(a*z)) + 1/b;
end
```

```
METODA NEWTON
function [w_final, errors, norms, times] = Newton(x, y, epsilon, maxIter)
    [n, N] = size(x);
    rng(0); % pentru reproductibilitate
    w = randn(n,1) * 0.01; % Initializare
    errors = zeros(1, maxIter);
    norms = zeros(1, maxIter);
    times = zeros(1, maxIter);

    iter = 0;
    tic;

    while norm(Grad(w, y, x)) > epsilon && iter < maxIter
        iter = iter + 1;

        g = Grad(w, y, x); % Gradient
        lambda = 1e-4; % regularizare mica
        H = hessiana(w, y, x);
        H = H + lambda * eye(size(H)); % regularizare
        w = w - H \ g'; % pas Newton


        errors(iter) = obj(w, y, x);
        norms(iter) = norm(g);
        times(iter) = toc;
```

```
    end

    errors = errors(1:iter);
    norms = norms(1:iter);
    times = times(1:iter);
    w_final = w;
end
```

```
function A = Grad(w, y, x)
    m = size(x, 2);
    f = zeros(m,1);
    for i = 1:m
        f(i) = sigmoid(w' * x(:,i));
    end
    A = ((f - y)' * x') / m;
end
```

```
function h = hessiana(w, y, x)
    m = size(x,2);
    n = size(x,1);
    z = zeros(m,1);
    for i = 1:m
        e = sigmoid(w' * x(:,i));
        z(i) = e * (1 - e);
    end
    Q = diag(z);
    h = (x * Q * x') / m;
end
```

```
function L = obj(w, y, x)
    m = size(x,2);
    f = zeros(m,1);
    for i = 1:m
        f(i) = sigmoid(w' * x(:,i));
    end
    eps = 1e-8; % protectie log(0)
    L = -(1/m) * sum(y .* log(f + eps) + (1 - y) .* log(1 - f + eps));
end
```