



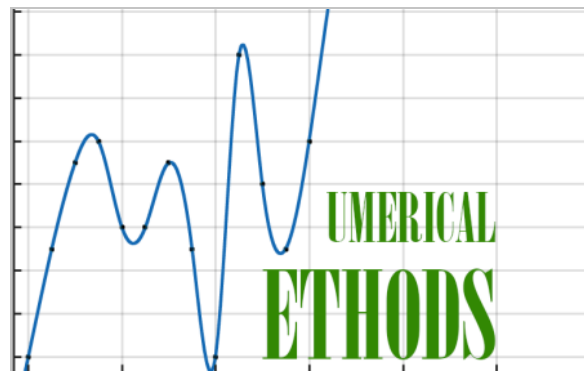
Tema1 Metode Numerice

rezolvarea temei la Metode Numerice - anul 1

Table of contents

Tema1 MN	2
Detectia Anomaliilor	4
estimate_gaussian	5
multivariate_gaussian	7
optimal_threshold	10
identify_outliers	14
Kernel Regression	16
kernel_functions	18
build_kernel	20
cholesky	22
get_lower_inverse	24
get_prediction_params	26
get_prediction_params_iter	28
split_dataset	31
Markov Text Generation	33
distinct_words	35
k_secv	36
distinct_k_secv	38
word_idx	39
k_secv_idx	41
stochastic_matrix	43

Tema1 MN



MN team

Motivatia temei

Prima temă de casă la Metode Numerice vizează următoarele obiective:

- Familiarizarea cu mediul de programare GNU Octave și facilitățile oferite de acesta;
- Folosirea matricelor și a sistemelor de ecuații liniare pentru a modela probleme reale, întâlnite în viața de zi cu zi, precum lanțurile Markov;
- Introducerea în învățarea supervizată.

Contents

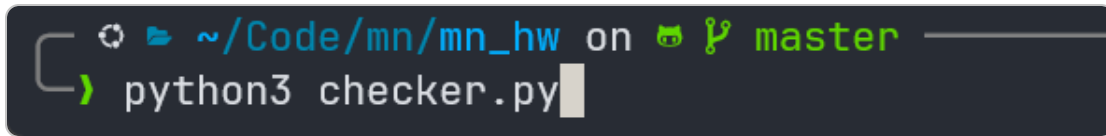
- Task 1: Detecția anomaliilor ([Detectia Anomaliilor](#))
- Task 2: Kernel Regression ([Kernel Regression](#))
- Task 3: Markov Text Generation ([Markov Text Generation](#))

Checker tema

Modul de folosire al checkerului:

Homework check

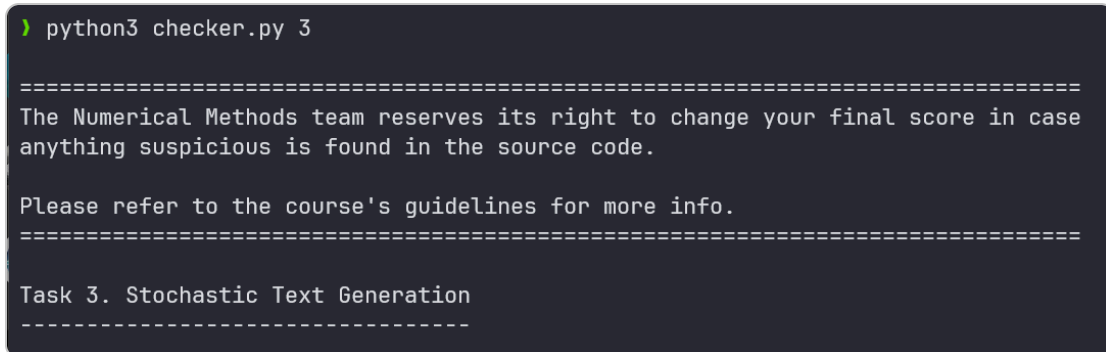
1. Din folderul temei, se executa cu `python3` fisierul `checker.py`.



```
~/Code/mn/mn_hw on master
python3 checker.py
```

completion suggestions for procedure

2. Se poate specifica ca argument si numarul taskului



```
python3 checker.py 3

=====
The Numerical Methods team reserves its right to change your final score in case
anything suspicious is found in the source code.

Please refer to the course's guidelines for more info.
=====

Task 3. Stochastic Text Generation
-----
```

completion suggestions for procedure

Author

Rares Andrei Sarماسag

student at Polithenica Bucuresti - Faculty of Automatic Control and Computer Science

Seria CB-312 @ CTI 2024

contact email: rares.sarmasag@stud.acs.upb.ro (<mailto:rares.sarmasag@stud.acs.upb.ro>).

discord: _ap0

See also

Tema1 MN documentation

My projects (<https://github.com/rares9301?tab=repositories>)

Github (<https://github.com/rares9301>)

Detectia Anomaliilor

Unul dintre primii pasi pe care trebuie sa ii faceti atunci cand vi se da un Training Data Set este sa detectati asa-zisele anomalii (outliers). Acest lucru poarta numele de Anomaly Detection si ajuta AI-ul caruia ii dati datele de training sa aiba un set de date consistent, fara greseli, si sa determine outlierii din datele de testing.

Functii

Au fost implementate urmatoarele functii:

estimate_gaussian

determina media si varianta pentru o distributie normala multivariata

multivariate_gaussian

calculeaza densitatea de probabilitate pentru distributia normala multivariata

optimal_threshold

determina cel mai bun factor ϵ

identify_outliers

determina outlierii (probabilitatile mai mici decat epsilon)

estimate_gaussian

Această funcție este folosită pentru a estima parametrii unei distribuții gaussiene (normale) pentru un set de date X

i Funcția nu depinde de alte funcții

Implementare

! function [mean_values, variances] = estimate_gaussian(X)

Funcția estimate_gaussian va returna două valori: mean_values și variances, care sunt vectorii mediei și matricea de covarianță

1. Calculul fiecărei caracteristici (coloane) din setul de date X . Rezultatul este un vector mean_vales care conține media pentru fiecare caracteristică.

```
mean_values = mean(X);
```

2. Calculul matricei de covarianță a caracteristicilor din X . Elementele de pe diagonala principală a matricei reprezintă varianțele fiecărei caracteristici.

```
variances = cov(X);
```

Cod sursa

```
function [mean_values, variances] = estimate_gaussian(X)

% Calculul mediei
mean_values = mean(X);

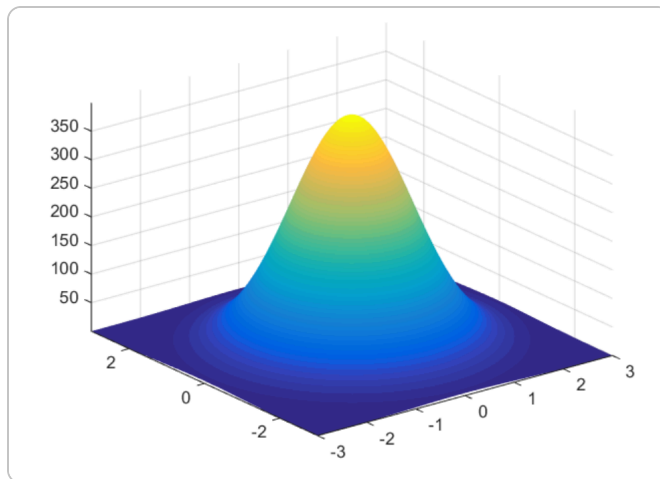
% Calculul matricei de varianta
variances = cov(X);
```

```
endfunction
```

multivariate_gaussian

Aceasta functie este folosita pentru a calcula probabilitățile unui set de date X sub o distribuție gaussiană multivariată, dată prin valorile medii `mean_values` și matricea de covarianță `variances`.

i Functia nu depinde de alte functii si este asemanatoare functiei `gaussian_distribution`. Pentru simplitate, voi explica doar functia `multivariate_gaussian`



multivariate_gaussian

Implementare

A `function probabilities = multivariate_gaussian(X, mean_values, variances)`

Funcția va returna un vector `probabilities` care conține probabilitățile pentru fiecare exemplu din X .

1. obțin dimensiunea matricei X , unde m reprezintă numărul de exemple și n numărul de caracteristici.

```
[m, n] = size(X);
```


2. calculez diferența dintre fiecare exemplu din X și vectorul mediei `mean_values`.

```
difference = bsxfun(@minus, X, mean_values)
```

3. calculez inversa matricei de covarianță `variances`.

```
inverse_variance = inv(variances);
```

4. calculez determinantul matricei de covarianță `variances`.

```
determinant = det(variances);
```

5. calculez exponentul pentru funcția de densitate a probabilității, care este o parte esențială a ecuației pentru distribuția gaussiană multivariată.

```
exponent = -0.5 * sum((difference * inverse_variance) .* difference, 2);
```

6. calculez probabilitățile pentru fiecare exemplu din X , folosind formula pentru distribuția gaussiană multivariată.

```
probabilities = (1 / (sqrt((2 * pi)^n * determinant))) * exp(exponent);
```

Cod sursa

```
function probabilities = multivariate_gaussian(X, mean_values,
variances)

[m, n] = size(X); % matrix size
difference = bsxfun(@minus, X, mean_values); % diff from mean
inverse_variance = inv(variances); % inverse of matrix
determinant = det(variances); % determinant

% exponent
exponent = -0.5 * sum((difference * inverse_variance) .* difference, 2);
```

```
% probabilitati  
probabilities = (1 / (sqrt((2 * pi)^n * determinant))) * exp(exponent);  
  
endfunction
```

optimal_threshold

Funcția este concepută pentru a găsi valoarea optimă a pragului (epsilon) care maximizează scorul F1 într-un context de clasificare binară.

⚠ Funcția apelează `check_predictions` și `metrics`

Prerequisites

Următoarele funcții sunt necesare pentru rezolvarea task-ului:

- `check_predictions`

```
function [false_positives, false_negatives, true_positives] =
    check_predictions(predictions, truths)

    % false positives
    false_positives = sum((predictions == 1) & (truths == 0));

    % false negatives
    false_negatives = sum((predictions == 0) & (truths == 1));

    % true positives
    true_positives = sum((predictions == 1) & (truths == 1));

    endfunction
```

- `metrics`


```
function [precision, recall, F1] = metrics(true_positives,
    false_positives, false_negatives)

    % precizie
    precision = true_positives / (true_positives + false_positives);

    % recall
    recall = true_positives / (true_positives + false_negatives);
```

```
% F1 score
F1 = 2 * (precision * recall) / (precision + recall);
endfunction
```

Implementare

 function [best_epsilon, best_F1, associated_precision, associated_recall] = optimal_threshold(truths, probabilities)

Funcția va returna valoarea optimă a pragului `best_epsilon`, cel mai bun scor F1 (`best_F1`), `associated_precision` și `associated_recall` asociate cu acest prag.

1. initializez rata de recall.

```
best_epsilon = 0;
best_F1 = 0;
associated_precision = 0;
associated_recall = 0;
```

2. setez `stepsize` și creez un vector `epsilons` cu aceste valori

```
stepsize = (max(probabilities) - min(probabilities)) / 1000;
epsilons = min(probabilities):stepsize:max(probabilities);
```

3. pentru fiecare valoare a lui `epsilon`, se generează un vector de predicții binare, unde probabilitățile mai mici decât `epsilon` sunt considerate pozitive (1), iar celelalte negative (0).

```
for epsilon = epsilons
    predictions = probabilities < epsilon;
```

4. calculez predicțiile.

```
[false_positives, false_negatives, true_positives] =
```

```
check_predictions(predictions, truths);
```

5. calcula precizia, rata de recall și scorul F1

```
[precision, recall, F1] = metrics(true_positives, false_positives,
false_negatives)
```

6. Dacă scorul F1 calculat pentru o anumită valoare a lui epsilon este mai mare decât cel mai bun scor F1 găsit până în acel moment, atunci valorile asociate cu acesta sunt actualizate ca fiind cele mai bune.

```
if F1 > best_F1
    best_F1 = F1;
    best_epsilon = epsilon;
    associated_precision = precision;
    associated_recall = recall;
end
```

Cod sursa

```
function [best_epsilon, best_F1, associated_precision,
associated_recall] = optimal_threshold(truths, probabilities)
    best_epsilon = 0;
    best_F1 = 0;
    associated_precision = 0;
    associated_recall = 0;

    stepsize = (max(probabilities) - min(probabilities)) / 1000;
    epsilons = min(probabilities):stepsize:max(probabilities);

    for epsilon = epsilons
        predictions = probabilities < epsilon;
        [false_positives, false_negatives, true_positives] =
check_predictions(predictions, truths);
        [precision, recall, F1] = metrics(true_positives, false_positives,
false_negatives)
```

```
    if F1 > best_F1
        best_F1 = F1;
        best_epsilon = epsilon;
        associated_precision = precision;
        associated_recall = recall;
    end
endfor
endfunction
```

identify_outliers

Funcția este utilizată pentru a identifica valorile **outliers** dintr-un set de date X , folosind valorile etichetate `yval`

⚠ Funcția apelează `estimate_gaussian`, `multivariate_gaussians` și `optimal_threshold`

Prerequisites

Următoarele funcții sunt necesare pentru rezolvarea task-ului:

- `estimate_gaussian` ([estimate_gaussian](#))
- `multivariate_gaussian` ([multivariate_gaussian](#))
- `optimal_threshold` ([optimal_threshold](#))

Implementare

⚠ `function [outlier_values, outlier_indices] = identify_outliers(X, yval)`

Funcția va returna valorile outliers (`outlier_values`) și indicii acestora (`outlier_indices`) în setul de date X .

1. calculez valorile medii și matricea de covarianță pentru setul de date X

```
[mean_values, covariance_matrix] = estimate_gaussian(X);
```

2. calculez probabilitățile fiecărui exemplu din X de a fi generat de o distribuție gaussiană multivariată,

```
probabilities = multivariate_gaussian(X, mean_values,  
covariance_matrix);
```

3. determin valoarea optimă a pragului (epsilon) care maximizează scorul F1, folosind valorile etichetate yval și probabilitățile calculate.

```
[epsilon, F1, precision, recall] = optimal_threshold(yval,
probabilities);
```

4. identific outliers.

```
outlier_indices = find(probabilities < epsilon);
```

Cod sursa

```
function [outlier_values, outlier_indices] = identify_outliers(X, yval)
    % valori medii si matricea de varianta
    [mean_values, covariance_matrix] = estimate_gaussian(X);

    % probabilitati folosind multivariate
    probabilities = multivariate_gaussian(X, mean_values,
covariance_matrix);

    % prag optim folosind scor F1
    [epsilon, F1, precision, recall] = optimal_threshold(yval,
probabilities);

    % outliers
    outlier_values = find(probabilities < epsilon);
endfunction
```


Kernel Regression

În viața de zi cu zi ne întâlnim cu ideea de predicție. De exemplu, ne dorim să estimăm cât de mult timp ne va consuma o temă, cât de mult timp vom sta în trafic, etc. Evident că răspunsurile la astfel de întrebări depind de foarte mulți parametri și ne este destul de greu să găsim relații de cauzalitate între ei.

Funcții

Au fost implementate următoarele funcții:

kernel_functions

funcții pentru a construi kernels de tipuri diferite

build_kernel

constructor pentru kernel

cholesky

metoda pentru descompunerea LU a unei matrici pozitiv semi-definite.

get_lower_inverse

inversează o matrice lower folosind un algoritm de eliminare gaussiană.

get_prediction-params

calculează parametrii de predicție

get_prediction-params-iter

calculeaza parametrii de predicție iterativ

split_dataset

imparte dataset in train_data si pred_data

kernel_functions

Rolul acestor kernel-uri este de a ne oferi o modalitate de a estima parametri necesari in functie de gradul maxim (din punct de vedere polinomial) pe care il atribuim functiei ϕ .

i Am implementat 3 tipuri de kernel-uri

Kernels

- Linear Kernel

$$K(x, y) = y^T * x$$

```
function retval = linear_kernel(x, y, other)
    % ignore 'other' parameter
    retval = dot(x, y);
endfunction
```

- Polynomial Kernel

$$K(x, y) = (1 + y^T * x)^d$$

```
function retval = polynomial_kernel (x, y, d)
    % kernel polynomial
    retval = (1 + dot(x, y))^d;
endfunction
```

- Gaussian/Radial - Basis Kernel

$$K(x, y) = \exp \left(-\frac{\|x - y\|_2^2}{2\sigma^2} \right)$$

```
function retval = gaussian_kernel (x, y, sigma)
    squared_distance = sum((x - y).^2);
```

```
retval = exp(-squared_distance / (2 * sigma^2));  
endfunction
```

build_kernel

Funcția este utilizată pentru a construi o matrice kernel, care este o componentă esențială în mulți algoritmi de învățare automată, inclusiv în SVMs.

i Funcția nu depinde de alte funcții

Implementare

! function [K] = build_kernel (X, f, f_param)

Funcția va returna o matrice kernel **K**, care este calculată folosind o funcție kernel **f** și un parametru al acestei funcții **f_param**

1. determin numărul de **num_data** din setul de date **X**.

```
num_data = size(X, 1);
```

2. inițializez matricea kernel **K** ca o matrice pătratică de dimensiune **num_data** x **num_data**

```
K = zeros(num_data, num_data);
```

3. parcurg fiecare pereche de exemple de date pentru a calcula valorile kernel.

```
for i = 1:num_data
    for j = 1:num_data
```

4. calculează valoarea kernel pentru perechea de exemple **i** și **j** folosind funcția kernel **f**

```
K(i, j) = f(X(i, :), X(j, :), f_param);
```

Cod sursa

```
function [K] = build_kernel (X, f, f_param)
    % set size of matrix
    num_data = size(X, 1);

    % memory allocate
    K = zeros(num_data, num_data);

    % compute kernel values
    for i = 1:num_data
        for j = 1:num_data
            % kernel value for i <-> j
            K(i, j) = f(X(i, :), X(j, :), f_param);
        end
    end
endfunction
```

cholesky

Descompunerea Cholesky este o metodă de factorizare a unei matrice simetrice pozitiv definite în produsul unei matrice inferioare triunghiulare și transpusa sa.

i Funcția nu depinde de alte funcții

Implementare

! function L = cholesky(A)

Funcția va returna o matrice triunghiulară inferioară **L** caracteristica descompunerii Cholesky

1. determin dimensiunea matricei **A** și initializez matricea **L** ca o matrice pătratică de dimensiune **n x n**

```
[n, m] = size(A);
L = zeros(n);
```

2. calculez elementele diagonale ale matricei **L** folosind rădăcina pătrată a diferenței dintre elementul corespunzător din **A** și suma pătratelor elementelor anterioare din aceeași linie.

```
for i = 1:n
    L(i, i) = sqrt(A(i, i) - dot(L(i, 1:i-1), L(i, 1:i-1)));
```

3. calculez elementele non-diagonale ale matricei **L** prin scăderea produsului scalar al elementelor anterioare din linia **i** și coloana **j** și împărțirea rezultatului la elementul diagonal curent din **L**

```
for j = i+1:n
    L(j, i) = (A(j, i) - dot(L(i, 1:i-1), L(j, 1:i-1))) / L(i, i);
```

Cod sursa

```
function L = cholesky(A)
    [n, m] = size(A);
    L = zeros(n);

    for i = 1:n
        L(i, i) = sqrt(A(i, i) - dot(L(i, 1:i-1), L(i, 1:i-1)));
        for j = i+1:n
            L(j, i) = (A(j, i) - dot(L(i, 1:i-1), L(j, 1:i-1))) / L(i, i);
        end
    end
endfunction
```


get_lower_inverse

Funcția este folosită pentru a calcula inversa unei matrice triunghiulare inferioare L .

i Funcția nu depinde de alte funcții

Implementare

A function P = get_lower_inverse(L)

Funcția va returna matricea P , care este inversa matricei triunghiulare inferioare L .

1. determin dimensiunea matricei L și se inițializează matricea P ca o matrice identitate de dimensiune $n \times n$.

```
n = size(L, 1);
P = eye(n);
```

2. calculez elementele diagonale ale matricei P prin inversarea elementelor diagonale ale matricei L .

```
for j = 1:n
    P(j, j) = 1 / L(j, j);
```

3. Calculez elementele sub diagonală 1 ale matricei P . Pentru fiecare element $P(i, j)$, se înmulțește secțiunea corespunzătoare a liniei i din L cu coloana j din P (până la elementul $i-1$), se negativizează rezultatul și se împarte la elementul diagonal $L(i, i)$.

```
for i = j+1:n
    P(i, j) = -L(i, j:i-1) * P(j:i-1, j) / L(i, i);
```

Cod sursa

```
function P = get_lower_inverse(L)
    n = size(L, 1);
    P = eye(n);
    for j = 1:n
        P(j, j) = 1 / L(j, j);
        for i = j+1:n
            P(i, j) = -L(i, j:i-1) * P(j:i-1, j) / L(i, i);
        end
    end
end
```

get_prediction_params

Funcția este folosită pentru a calcula parametrii de predicție pentru un model de învățare automată, dată o matrice kernel `K`, un vector de etichete `y` și un parametru de regularizare `lambda`.

⚠ Funcția apelează funcția `cholesky`

Prerequisites

Urmatoarele functii sunt necesare pentru rezolvarea task-ului:

- `cholesky` ([cholesky](#))

Implementare

⚠ function [a] = get_prediction_params (K, y, lambda)

Funcția va returna un vector `a` care conține parametrii de predicție.

1. determin dimensiunea matricei kernel `K`

```
n = size(K, 1);
```

2. Se adaugă regularizare la matricea kernel `K` prin adăugarea produsului dintre `lambda` și matricea identitate `eye(n)` la `K`. Apoi se aplică descompunerea Cholesky pentru a obține o matrice triunghiulară inferioară `L`.

```
L = cholesky(K + lambda * eye(n));
```

3. calculez inversa matricei triunghiulare inferioare `L`.

```
Linv = get_lower_inverse(L);
```

4. rezolv sistemul liniar pentru a obține parametrii de predicție a . Acest lucru se face prin înmulțirea transpusei matricei inverse L_{inv}' cu rezultatul înmulțirii matricei inverse L_{inv} cu vectorul de etichete y .

```
a = Linv' * (Linv * y);
```

Cod sursa

```
function [a] = get_prediction_params (K, y, lambda)
    n = size(K, 1);
    L = chol(K + lambda * eye(n)); % cholesky
    Linv = get_lower_inverse(L); % inverse
    a = Linv' * (Linv * y); % rezolv sistemul liniar
endfunction
```

get_prediction_params_iter

Funcția este folosită pentru a calcula parametrii de predicție într-un mod iterativ, folosind metoda gradientului conjugat.

⚠ Funcția apelează `conjugate_gradient`

Prerequisites

Urmatoarele functii sunt necesare pentru rezolvarea task-ului:

- `conjugate_gradient`


```
function [x] = conjugate_gradient_optimized(A, b, x0, tol, max_iter)
    % initializez variabile
    r = b - A * x0; % reziduu
    v = r;          % directie de cautare
    x = x0;         % solutie
    tol_sq = tol^2; % toleranta^2 (eficientizare)
    k = 0;          % contor

    % pre-calculez prod scalar al reziduului initial
    rho = r' * r;

    % Iterez pana la convergenta sau max_iter
    while k < max_iter && rho > tol_sq
        Av = A * v;
        alpha = rho / (v' * Av);
        x = x + alpha * v; % actualizez solutia
        r = r - alpha * Av; % actualizez reziduu
        rho_new = r' * r; % noul produs scalar al reziduului
        beta = rho_new / rho;
        v = r + beta * v; % actualizez directia de cautare
        rho = rho_new; % actualizez produsul scalar al reziduului
        k = k + 1; % incrementez contor
```

```
end
end
```

Implementare

 function [a] = get_prediction_params_iterative (K, y, lambda)

Funcția va returna un vector `a` care conține parametrii de predicție calculați.

1. determin numărul de linii din matricea kernel `K`

```
m = size(K, 1);
```

2. regularizez matricea kernel `K` prin adăugarea produsului dintre `lambda` și matricea identitate `eye(m)` la `K`, rezultând matricea `A`

```
A = lambda * eye(m) + K;
```

3. inițializez vectorul de start `x0` pentru algoritmul iterativ, setând toate valorile la zero.

```
x0 = zeros(m, 1);
```

4. setez toleranta

```
tol = 1e-6;
```

5. setez un număr maxim de iterații `max_iter` pentru algoritmul iterativ.

```
max_iter = 100;
```

6. aplic metoda gradientului conjugat pentru a rezolva sistemul liniar

```
a = conjugate_gradient(A, y, x0, tol, max_iter);
```

Cod sursa

```
function [a] = get_prediction_params_iterative (K, y, lambda)
    % numarul de linii
    m = size(K, 1);

    % regularizez matricea K
    A = lambda * eye(m) + K;

    % initializez x0
    x0 = zeros(m, 1);

    % Setez toleranta
    tol = 1e-6;

    % numarul maxim de iteratii
    max_iter = 100;

    % aplic conjugatul
    a = conjugate_gradient(A, y, x0, tol, max_iter);
endfunction
```

split_dataset

Funcția este folosită pentru a împărți un set de date `X` și etichetele asociate `y` în două subgrupuri: unul pentru antrenament și unul pentru predicții, bazat pe un procentaj dat.

i Funcția nu depinde de alte funcții

Implementare

! `function [X_train, y_train, X_pred, y_pred] = split_dataset (X, y, percentage)`

Funcția va returna patru seturi de date: `X_train` și `y_train` pentru antrenament, și `X_pred` și `y_pred` pentru predicții.

1. determin numărul total de puncte de date din setul `X`

```
num_data = size(X, 1);
```

2. calculez numărul de puncte de date care vor fi folosite pentru antrenament, rotunjind produsul dintre procentajul dat și numărul total de date.

```
num_train_data = round(percentaje * num_data);
```

3. creez un vector de indici `train_indices` pentru datele de antrenament

```
train_indices = 1:num_train_data;
```

4. creez un vector de indici `pred_indices` pentru datele de predicție

```
pred_indices = num_train_data + 1:num_data;
```

5. extrag datele de antrenament `X_train` și etichetele `y_train` folosind indicii de antrenament.


```
X_train = X(train_indices, :);
y_train = y(train_indices);
```

6. extrag datele de predicție `X_pred` și etichetele `y_pred` folosind indicii de predicție.

```
X_pred = X(pred_indices, :);
y_pred = y(pred_indices);
```

Cod sursa

```
function [X_train, y_train, X_pred, y_pred] = split_dataset (X, y,
percentage)
    % total data points
    num_data = size(X, 1);

    % round
    num_train_data = round(percentage * num_data);

    % indices for training
    train_indices = 1:num_train_data;

    % indices for preds
    pred_indices = num_train_data + 1:num_data;

    % training data
    X_train = X(train_indices, :);
    y_train = y(train_indices);

    % prediction data
    X_pred = X(pred_indices, :);
    y_pred = y(pred_indices);

endfunction
```

Markov Text Generation

Cel mai simplu task actual, in LLMs este Text Generation. ChatGPT insusi este un "text generator" implementat ca Transformer. In acest task, nu vom ajunge nici pe departe la nivelul lui ChatGPT, ci vom incerca insa sa cream un mini-AI bazat pe Lanturi Markov care poate genera text care sa semene cu un fisier de input. Scopul acestui task este in primul rand sa va familiarizeze cu lucrul cu Stringuri si Cells in MATLAB, si mai apoi sa va prezinte terminologia folosita in AI training.

Functii

Au fost implementate urmatoarele functii:

distinct_words

intoarce tokenurile sortate si unice.

k_secv

pentru un cell-array unidimensional returneaza un cell-array de k-secv

distinct_k_secv

intoarce k-secvurile sortate si unice.

word_idx

intoarce un dictionar care contine indecsii asociati fiecarui label.

k_secv_idx

intoarce un dictionar care contine indecsii asociati fiecarui feature (k-secv).

stochastic_matrix

creeaza matricea stochastica

distinct_words

Funcția este folosită pentru a returna o listă de cuvinte unice dintr-un vector de tokeni `tokens`.

i Funcția nu depinde de alte funcții

Implementare

! function retval = distinct_words(tokens)

Funcția va returna un vector `retval` care conține tokenii unici.

- sortez vectorul de tokeni `tokens` și apoi aplic funcția `unique` pentru a elimina duplicatele, rezultând într-un vector de tokeni unici `retval`.

```
retval = unique(sort(tokens));
```

Cod sursa

```
function retval = distinct_words(tokens)
    % sort unique
    retval = unique(sort(tokens));
endfunction
```

k_secv

Funcția este folosită pentru a genera toate secvențele posibile de lungime k dintr-un vector de cuvinte A .

i Funcția nu depinde de alte funcții

Implementare

! function B = k_secv (A, k)

Funcția va returna un cell array B care conține secvențele de cuvinte.

1. Se iterează prin vectorul de cuvinte A , de la primul cuvânt până la $\text{length}(A)-k$. Se creează o secvență de k cuvinte și se adaugă la cell array-ul B .

```
for i = 1:length(A)-k
    B{end+1} = strjoin(A(i:i+k-1), ' ');
endfor
```

2. transpun B pentru a avea un format coloană.

```
B = B';
```

Cod sursa

```
function B = k_secv (A, k)
    B = {};
    for i = 1:length(A)-k
        B{end+1} = strjoin(A(i:i+k-1), ' ');
    endfor
```

```
B = B';  
endfunction
```

distinct_k_secv

Funcția este folosită pentru a elimina duplicatele dintr-un cell array `cell_array`, lăsând doar elementele unice.

i Funcția nu depinde de alte funcții

Implementare

! function unique_cell_array = distinct_k_secv (cell_array)

Funcția va returna un cell array `unique_cell_array` care conține elementele unice din `cell_array`.

1. aplic funcția `unique` direct pe `cell_array` pentru a obține elementele unice, care sunt apoi stocate în `unique_cell_array`.

```
unique_cell_array = unique(cell_array);
```

Cod sursa

```
function unique_cell_array = distinct_k_secv (cell_array)
    % unique
    unique_cell_array = unique(cell_array);
endfunction
```

word_idx

Funcția este folosită pentru a crea un dicționar care asociază fiecare cuvânt unic dintr-un array cu un indice unic.

i Funcția nu depinde de alte funcții

Implementare

! function retval = word_idx (distinct_wds)

Funcția va returna un dicționar `retval` care mapează cuvintele la indici.

1. creez un array de celule indices care conține numerele de la 1 la lungimea array-ului `distinct_wds`, fiecare număr fiind convertit într-o celulă.

```
indices = num2cell(1:length(distinct_wds));
```

2. creez un dicționar `retval` folosind obiectul `containers.Map`. Cuvintele unice din `distinct_wds` sunt folosite ca *chei*, iar indicii corespunzători din `indices` sunt folosiți ca *valori*.

```
retval = containers.Map(distinct_wds, indices);
```

Cod sursa

```
function retval = word_idx (distinct_wds)
    % array de indici 1 -> len distinct_wds
    indices = num2cell(1:length(distinct_wds));

    % dictionar cu cheile distinct_wds
```



```
    retval = containers.Map(distinct_wds, indices);  
endfunction
```

k_secv_idx

Funcția este folosită pentru a crea un dicționar care asociază fiecare secvență distinctă de cuvinte dintr-un array cu un indice unic.

i Funcția nu depinde de alte funcții

Implementare

! function retval = k_secv_idx (distinct_k_sec)

Funcția va returna un dicționar `retval` care mapează secvențele de cuvinte la indici.

1. creez o listă de indici `indices` care conține numerele de la 1 până la lungimea array-ului `distinct_k_sec`.

```
indices = 1:length(distinct_k_sec);
```

2. creez un dicționar `retval` folosind obiectul `containers.Map`. Secvențele unice din `distinct_k_sec` sunt folosite ca *chei*, iar indicii corespunzători din `indices` sunt folosiți ca *valori*.

```
retval = containers.Map(distinct_k_sec, indices);
```

Cod sursa

```
function retval = k_secv_idx (distinct_k_sec)
    % listă de indici 1 la len distinct_k_sec
    indices = 1:length(distinct_k_sec);

    % dicționar cu cheile distinct_k_sec
```

```
    retval = containers.Map(distinct_k_sec, indices);  
endfunction
```

stochastic_matrix

Funcția este folosită pentru a construi o matrice stocastică dintr-un corpus de cuvinte, bazată pe secvențe de cuvinte de lungime `k`.

i Funcția nu depinde de alte funcții

Implementare

! function retval = stochastic_matrix(k_secv_corpus, corpus_words, words_set, k_secv_set, k)

Funcția va returna o matrice stocastică `retval`, care reprezintă frecvențele de tranziție între secvențele de cuvinte și cuvintele următoare din corpus.

1. convertesc cuvintele din corpus și secvențele de cuvinte în seturi unice pentru a reduce redundanța și pentru a accelera accesul la date.

```
[words_set, ~, corpus_idx] = unique(corpus_words);
[~, ~, k_secv_idx] = unique(k_secv_corpus);
```

2. prealoc matricea de rezultate `retval` cu zerouri,

```
retval = zeros(length(k_secv_set), length(words_set));
```

3. calculează indexul maxim până la care se poate itera în corpus fără a depăși limita acestuia

```
max_word_index = length(corpus_words) - k;
```

4. iterez prin corpus și se calculez frecvențele de apariție a fiecărui cuvânt care urmează după o secvență de cuvinte de lungime `k`. Frecvențele sunt actualizate în matricea `retval`.

```

for i = 1:max_word_index
    k_seq_index = k_secv_idx(i);
    next_word_index = corpus_idx(i + k);
    retval(k_seq_index, next_word_index) = retval(k_seq_index,
next_word_index) + 1;
end

```

Cod sursa

```

function retval = stochastic_matrix(k_secv_corpus, corpus_words,
words_set, k_secv_set, k)
    % Convertesc seturile in celule pentru acces ultra fast
    [words_set, ~, corpus_idx] = unique(corpus_words);
    [~, ~, k_secv_idx] = unique(k_secv_corpus);

    % prealloc matricea de rezultate
    retval = zeros(length(k_secv_set), length(words_set));

    % Calculez index maxim
    max_word_index = length(corpus_words) - k;

    % vectorizez si calculez frecventele
    for i = 1:max_word_index
        k_seq_index = k_secv_idx(i);
        next_word_index = corpus_idx(i + k);
        retval(k_seq_index, next_word_index) = retval(k_seq_index,
next_word_index) + 1;
    end
endfunction

```