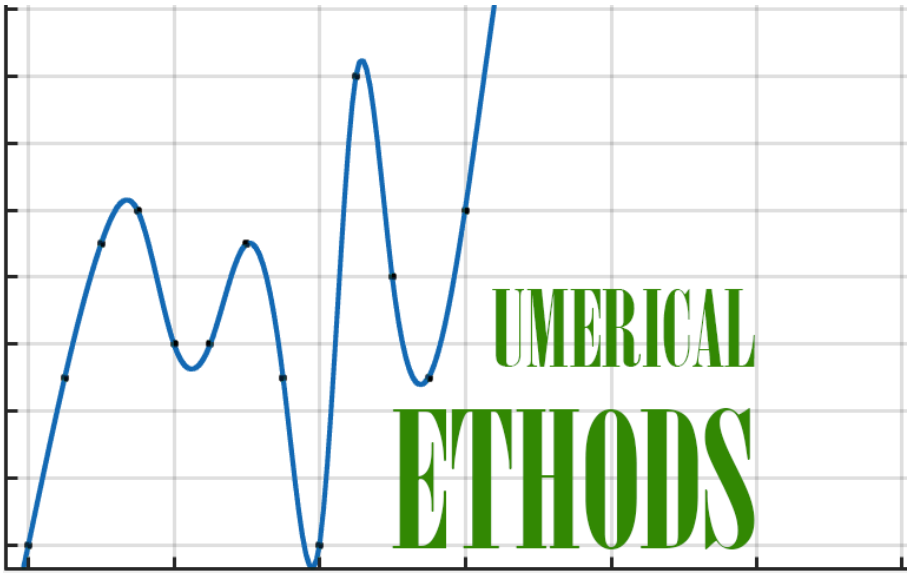
Metode Numerice

Tema de casă 1 - Analiza datelor

Deadline Soft: 25 Aprilie 2024 | *Deadline Hard*: 30 Aprilie 2024



Autori: Adrian-Nicolae ARITON, Horia MERCAN,
Alexandru-Constantin ARITON, Carol-Luca GASAN, Andrei STAN

Obiectivele temei de casă

Prima temă de casă la Metode Numerice vizează următoarele aspecte:

- Familiarizarea cu limbajul de programare MATLAB și cu mediul de programare GNU Octave, precum și cu facilitățile oferite de acesta;
- Folosirea matricelor și a sistemelor de ecuații liniare pentru a modela probleme reale, oferind o introducere de bază în metodele numerice folosite în domeniul învățării automate (en: *Machine Learning*);
- O introducere în metoda de învățarea supervizată (en: *Supervised Machine Learning*);
- Implementarea unor biblioteci de funcții în MATLAB și testarea funcțiilor componente pe diferite seturi de date.

Notă: Metodele prezentate în cadrul acestei teme de casă au o fundamentare matematică. Echipa temei a prezentat în detaliu aspectele practice și de implementare, pentru atingerea obiectivelor temei de casă. Acolo unde a fost cazul, au fost indicate resurse suplimentare, pentru cei care doresc aprofundarea metodelor.

Cuprins

1 Metoda 1: Detecția anomaliilor

1.1 Introducere

Analiza seturilor de date folosite pentru antrenare (en: *Training Dataset*) începe cu un prim pas prin care se detectează anomaliile (en: *outliers*). Intuitiv, o anomalie în setul de date poate fi înțeleasă ca un subset al datelor de antrenare care nu se potrivește cu întreg setul de date. Scopul primei metode pe care o vom proiecta și implementa este să înțelegem ce înseamnă această (ne)potrivire.

Această metodă se numește detecția anomaliilor (en: *Anomaly Detection*) și este necesară și utilă în cazul modelelor care folosesc datele de antrenare și doresc să aibă un set de date consistent, fără greșeli, și să determine *anomaliile* din datele de antrenare / testare.

1.2 Prezentarea metodei

Setul de date folosit pentru analiză

Într-un set de date se dau mai multe elemente de forma $\left\{x_{(1)}, x_{(2)}, \ldots, x_{(m)}\right\}$. Fiecare element este un vector de dimensiune n, $x_{(i)} \in \mathbb{R}^n$. Pentru acest set de date (en: *dataset*) definim vectorul medie $\mu \in \mathbb{R}^n$, pe componente, astfel:

$$\mu(i) = \frac{1}{m} \sum_{k=1}^{m} x_{(k)}(i), \quad i = 1..n,$$

precum și o matrice specifică numită *matricea de varianță*:

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x_{(i)} - \mu\right) \left(x_{(i)} - \mu\right)^T, \quad \Sigma \in \mathbb{R}^{n \times n}$$

Definim funcția care descrie probabilitatea de apariție a unui element $x_{(i)}$, printre valorile din dataset-ul folosit în analiză, ca fiind:

$$f(x_{(i)}) = \frac{1}{(2\pi)^{\frac{n}{2}} \det(\Sigma)^{\frac{1}{2}}} \exp\left(-\frac{1}{2} \left(x_{(i)} - \mu\right)^T \Sigma^{-1} \left(x_{(i)} - \mu\right)\right).$$

Definiția unei anomalii

Un vector $x_{(i)}$ este o *anomalie* față de setul de date considerat, dacă pentru acel vector funcția f definită este mai mică decât un anumit prag ε, concret $f(x_{(i)}) < \varepsilon$. Denumim ε pragul optim care determină toate anomaliile din dataset-ul considerat.

Determinarea outlier-ilor (Anomaly detection)

Determinarea outlier-ilor dintr-un set de date constă în trei etape, după cum urmează:

1. Calcularea mediei μ și a matricii de varianță Σ pentru dataset-ul considerat și calcularea funcției f pentru toate elementele din cadrul dataset-ului, $f(x_{(i)})$.

2. Estimarea factorului ε, din setul de antrenament. Pentru fiecare ε încercat calculăm parametri:

   - **precision**, adică procentajul outlierilor adevărați și determinați (true positives) din outlierii determinați pentru acel ε (total positives);
   - **recall**, adică procentajul outlierilor adevărați și determinați (true positives) din totalul outlierilor reali (true positives + false negatives);
   - **F1**, parametrul ce trebuie să fie minim pentru cel mai bun ε, definit astfel:

$$\mathbf{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

3. După alegerea lui $\varepsilon_{\text{best}}$ astfel încât parametrul **F1** să fie minim, determinăm toți outlierii din testing dataset ca fiind toți vectorii $x_{(k)}$ pentru care $f(x_{(k)}) < \varepsilon_{\text{best}}$.

1.3 Funcțiile din cadrul bibliotecii

Această metodă constă în implementarea celor trei pași pentru determinarea celui mai bun threshold (prag), precum și în determinarea vectorului μ și a matricei Σ pentru un anumit dataset.

```
function [mean_values variances] = estimate_gaussian(X)
% Aceasta functie va determina media si varianta pentru datasetul dat.
```

```
function probability = gaussian_distribution(X, mean_value, variance)
% Aceasta functie calculeaza densitatea de probabilitate pentru datasetul dat.
```

```
function [false_pozitives, false_negatives, true_pozitives] =
check_predictions(predictions, truths)
% Aceasta functie este utilizata pentru determinarea pozitivelor false si adevarate
    precum si a negativelor false.
```

```
function [best_epsilon best_F1 associated_precision associated_recall] =
optimal_threshold(truths, probabilities)
% Aceasta functie este utilizata pentru determinarea celui mai bun factor 'epsilon'.
    Ulterior, outlierii vor fi toti cei pentru care probabilitatea este mai mica
    decat 'epsilon'.
```

2 Metoda 2: Kernel Regression

2.1 Introducere

În viața de zi cu zi ne întâlnim cu ideea de predicție (estimare) pentru diverse activități sau fenomene. De exemplu, ne dorim să estimăm cât de mult timp ne va consuma o temă sau cât de mult timp vom sta în trafic, dacă va ploua sau nu la final de săptămână, *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.

Astfel, pentru seturi de date foarte mari și pentru generarea unor predicții cât mai precise avem nevoie de diferite modele de Învățare Automată (en: *Machine Learning*, ML).

Pentru a doua sarcină de lucru propusă veți avea de implementat metodele numerice asociate unui algoritm de *Supervised Machine Learning* denumit *Kernel Regression*. În semestrul I, la ALGAED, cu siguranță ați discutat despre dreapta de regresie (ca în exemplul grafic din Figura 1) ca dreapta ce stabilește cea mai potrivită dependență liniară.

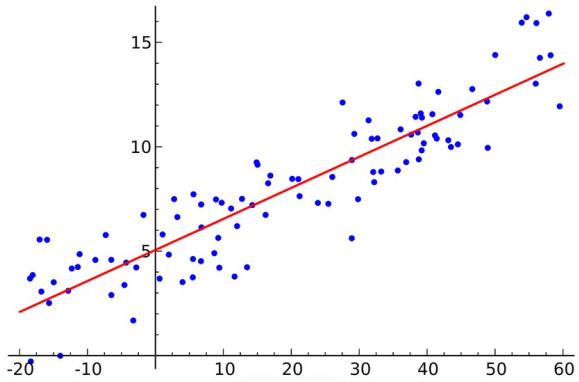


Figure 1: Dreapta de regresie ce trece printr-un set de date (en: *Linear Regression*).

Totuși, majoritatea fenomenelor pe care le avem de prelucrat nu sunt neapărat liniare. În multe cazuri încercăm să ne "liniarizăm" datele pentru a reuși să aplicăm diferite metode numerice matriceale pe acestea.

2.2 Cadru teoretic & Explicații

Regresia reprezintă, în esență, găsirea unor legături între anumite date. Aceasta se reduce la minimizarea unei funcții de cost și a pierderilor asociate (vom explica mai detaliat în paragrafele următoare aceste concepte).

Să presupunem că avem mai multe perechi de forma $(\boldsymbol{x}^{(i)}, y^{(i)})$, unde $\boldsymbol{x}^{(i)}$ reprezintă datele de intrare cărora li se asociază valoarea de ieșire $y^{(i)}$, cu $\boldsymbol{x}^{(i)} \in \mathbb{R}^k$ și $y^{(i)} \in \mathbb{R}$. Am menționat că pentru acest tip de regresie avem nevoie de o funcție care să reprezinte această liniarizare a datelor $\boldsymbol{\phi} : \mathbb{R}^k \to \mathbb{R}^n$.

De exemplu, dacă avem un set de puncte 2D pe o parabolă ce trece prin origine, fiind simetrică față de axa Oy, o astfel de funcție ar putea fi $\boldsymbol{\phi}(x) = x^2$ și astfel funcția $y(x^2)$ ar putea reprezenta o dependență liniară între seturile de puncte $y^{(i)}$ și $\boldsymbol{x}^{(i)}$ (evident că în acest exemplu s-a considerat $k = 1$ și $n = 1$).

Obiectivul nostru este determinarea coeficienților specifici (în engleză se numesc ***weigths***) care să estimeze cât mai bine ieșirea (funcția y) pentru un set de date de intrare dat. Din punct de vedere numeric, aceasta s-ar traduce în determinarea unui vector $\theta \in \mathbb{R}^n$ din care să rezulte o predicție de forma $y_{pred} = \theta^T \phi(\boldsymbol{x})$.

Asemenea dreptei de regresie care minimizează pătratul erorii dintre valoarea reala și predicție, și noi vom alege o funcție de cost similară:

$$J_{cost}(\theta) = \sum_{i=1}^{m} (y^{(i)} - \theta^T \phi(\boldsymbol{x}^{(i)}))^2 + \lambda * \|\theta\|_2^2 ,$$

unde:

- $\|\theta\|_2$ reprezintă norma 2 (euclideană) a coeficienților modelului, adică $\|\theta\|_2^2 = \sum_{i=1}^{n} |\theta_i|^2$.
- $\lambda \in \mathbb{R}_+$ este **parametrul** care controlează regularizarea (în metodele de ML acest parametru este important pentru o evaluare cât mai buna a modelului și pentru a nu apărea anomalii datorate bias-ului setului de date. În soluția propusă de voi trebuie să țineți cont de el în funcțiile pe care le aveți de implementat, dar noi vă vom oferi acest parametru în teste, nu îl aveți de evaluat/determinat).

Vom prelucra această funcție de cost, astfel încât să obținem minimul dorit. Vom începe prelucrarea prin a pune condiția necesară asupra gradientului funcției (echivalentul derivatei unei funcții) să fie nul.

$$\frac{dJ}{d\theta} = -2 \sum_{i=1}^{m} [(y^{(i)} - \theta^T \phi(\boldsymbol{x}^{(i)}))\phi(\boldsymbol{x}^{(i)})] + 2\lambda\theta = 0.$$

Vom folosi următoarea notație, $\alpha_i = y^{(i)} - \theta^T \phi(\boldsymbol{x}^{(i)})$, pentru a exprima vectorul θ optim:

$$\theta = \frac{1}{\lambda} \sum_{i=1}^{m} \alpha_i \phi(\boldsymbol{x}^{(i)}).$$

Ca observație, θ optim trebuie să fie o combinație liniară a vectorilor de intrare (input) cărora li se aplică funcția de "liniarizare" datorată regularizării pe care am adăugat-o. Scopul nostru acum este determinarea parametrilor α_i $(i = 1, ..., m)$:

$$\alpha_i = y^{(i)} - \theta^T \phi(\boldsymbol{x}^{(i)}) = y^{(i)} - \frac{1}{\lambda} \sum_{j=1}^{m} \alpha_j \phi(\boldsymbol{x}^{(j)})^T \phi(\boldsymbol{x}^{(i)}).$$

Astfel, ajungem la concluzia că α_i depinde într-un mod liniar doar de produsele scalare dintre datele de intrare cărora li se aplică o funcție ϕ și răspunsurile inițiale.

Considerăm notația $K(x^{(i)}, x^{(j)}) = \phi(\boldsymbol{x}^{(j)})^T \phi(\boldsymbol{x}^{(i)})$ și vom denumi $K : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ funcție de kernel. Această funcție reprezintă un produs scalar peste $\mathbb{R}^k$.

Vom construi matricea Gram [1] astfel:

$$\boldsymbol{K} = \begin{bmatrix} \phi(\boldsymbol{x}^{(1)})^T\phi(\boldsymbol{x}^{(1)}) & \dots & \phi(\boldsymbol{x}^{(1)})^T\phi(\boldsymbol{x}^{(m)}) \\ \phi(\boldsymbol{x}^{(2)})^T\phi(\boldsymbol{x}^{(m)}) & \dots & \phi(\boldsymbol{x}^{(2)})^T\phi(\boldsymbol{x}^{(m)}) \\ \vdots & \ddots & \vdots \\ \phi(\boldsymbol{x}^{(m)})^T\phi(\boldsymbol{x}^{(1)}) & \dots & \phi(\boldsymbol{x}^{(m)})^T\phi(\boldsymbol{x}^{(m)}) \end{bmatrix}$$

Așa cum s-a prezentat în cadrul cursului de Metode Numerice, vă reamintim că această matrice este pozitiv semi-definită. Vom scrie egalitățile de mai sus pentru termenii α_i în forma matriceală astfel:

$$\boldsymbol{a} = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \end{bmatrix}^T, \quad \boldsymbol{y} = \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m)} \end{bmatrix}^T,$$

$$\boldsymbol{a} = \boldsymbol{y} - \frac{1}{\lambda}\boldsymbol{K}\boldsymbol{a}.$$

Ultima ecuație este echivalentă în cele din urmă cu:

$$\boldsymbol{a} = \lambda(\lambda \boldsymbol{I_m} + \boldsymbol{K})^{-1}\boldsymbol{y}$$

Practic, pentru găsirea vectorului $\boldsymbol{a}$ avem de inversat o matrice $m \times\ m$, pozitiv semi-definită. În cadrul acestei etape veți avea de făcut acest lucru în doua moduri distincte, folosindu-vă atât de metode clasice (învățate în cadrul laboratorului de Metode Numerice) cât și de o metodă iterativă optimă în astfel de cazuri (Metoda Gradientului Conjugat).

În cadrul algoritmilor de Kernel Regression nu avem nevoie propriu zis de estimarea θ așa cum era descris inițial (ci doar de vectorul $\boldsymbol{a}$) pentru predicție, întrucât pentru un input oarecare $\boldsymbol{x}$, predicția noastră are forma:

$$y_{pred} = \theta^T\phi(\boldsymbol{x}) = \frac{1}{\lambda}\sum_{i=1}^{m}\alpha_i\phi(\boldsymbol{x}^{(i)})^T\phi(\boldsymbol{x}) = \frac{1}{\lambda}\sum_{i=1}^{m}\alpha_i K(\boldsymbol{x}^{(i)}, \boldsymbol{x}).$$

Astfel, din punct de vedere practic, noi nu avem nevoie de funcțiile ϕ, ci doar de o funcție de kernel care să calculeze un produs scalar favorabil de la un caz la altul (uneori, pentru modelarea matematică a unor experimente funcția ϕ are o dimensiune infinită, însă produsul scalar asociat se poate calcula într-un mod bine-determinat și finit).

2.3 Tipuri de kernel

Rolul acestor kernel-uri este de a ne oferi o modalitate de a estima parametri necesari în funcție de gradul maxim (din punct de vedere polinomial) pe care îl atribuim funcției ϕ. Astfel, în cadrul acestui task vom defini următoarele tipuri de kernel, a căror implementare o veți avea de realizat:

- ***Linear Kernel***: $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{y}^T\boldsymbol{x}$;
- ***Polynomial Kernel***: $K(\boldsymbol{x}, \boldsymbol{y}) = (1+\boldsymbol{y}^T\boldsymbol{x})^d$, unde d este dimensiunea maximă a funcției polinomiale pe care o avem de aproximat;
- ***Gaussian/Radial − Basis Kernel***: $K(\boldsymbol{x}, \boldsymbol{y}) = exp(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|_2^2}{\sigma^2})$.

[1] http://www.math.clemson.edu/~macaule/classes/f20_math8530/slides/math8530_lecture-7-03_h.pdf

2.4 Metoda Gradientului Conjugat

În practică, există metode iterative deterministe pentru rezolvarea ecuațiilor liniare de forma $Ax = b$, unde A este o matrice simetrică pozitiv semi-definită. O astfel de metodă numerică este *metoda gradientului conjugat*[2] care face uz de construcția unor vectori A-ortogonali care se află în $Span < b, Ab, A^2b, \ldots, A^{(m-1)}b >$ (unde m este dimensiunea spațiului vectorial) și de determinarea exactă a soluției în m pași iterativi. Astfel, inversarea unei astfel de matrici poate fi mai rapidă. Reamintim algoritmul gradientului conjugat:

Conjugate Gradient Method

procedure CONJUGATE_GRADIENT(A, b, x_0, tol, max_iter)
  $r^{(0)} \leftarrow b - Ax_0$
  $v^{(1)} \leftarrow r^{(0)}$
  $x \leftarrow x_0$
  $tol_{squared} \leftarrow tol^2$
  $k \leftarrow 1$
  while $k <= max\_iter$ and ${r^{(\text{k-1})}}^T r^{(\text{k-1})} > tol_{squared}$ **do**
    $t_k \leftarrow \frac{{r^{(\text{k-1})}}^T r^{(\text{k - 1})}}{{v^{\text{k}}}^T A v^{\text{k}}}$
    $x^{(\text{k})} \leftarrow x^{(\text{k-1})} + t_k v^{(\text{k})}$
    $r^{(\text{k})} \leftarrow r^{(\text{k-1})} - t_k A v^{(\text{k})}$
    $s_k \leftarrow \frac{{r^{(\text{k})}}^T r^{(\text{k})}}{{r^{(\text{k-1})}}^T r^{(\text{k-1})}}$
    $v^{(\text{k+1})} \leftarrow r^{(\text{k})} + s_k v^{(\text{k})}$
    $k \leftarrow k + 1$
  return x

2.5 Funcțiile din cadrul bibliotecii

Pentru implementarea metodei prezentate vă propunem următorul set de funcții. Primele 3 funcții sunt reprezentate prin funcțiile de kernel, așa cum sunt explicate mai sus. Pentru simplitate acestea primesc 3 parametri (la kernelul liniar puteți ignora al 3-lea parametru în logica funcției). Doar pentru aceste 3 funcții aveți în vedere primii doi parametri că fiind vectori linie n-dimensionali.

```
function retval = linear_kernel (x, y, other)
% Aceasta este utilizata pentru implementarea functiei pentru kernelul liniar.
```

```
function retval = polynomial_kernel (x, y, d)
% Aceasta este utilizata pentru implementarea functie pentru kernelul polinomial.
```

```
function retval = gaussian_kernel (x, y, sigma)
% Aceasta este utilizata pentru implementarea functie pentru kernelul gausian.
```

[2] https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

```
function [X_train, y_train, X_pred, y_pred] = split_dataset (X, y, percentage)
% Aceasta functie divizeaza setul de date pentru antrenare si testare. Functia
    primeste ca parametru o matrice X ale carei linii reprezinta vectorii x^(i), un
    vector coloana y care reprezinta valorile asociate acestor puncte. Dat fiind 0 <
    percentage < 1, construiti perechile (X_train, y_train) si (X_pred, y_pred) care
    reprezinta setul de date pe care va fi antrenat modelul, respectiv setul de
    puncte a caror valoare o aveti de prezis si va fi verificata cu valorile reale.
    Alcatuiti perechile folosind primele percentage * numarul_total_de_date linii,
    respectiv ultimele (1 - percentage) * numarul_total_de_date linii. Folositi round.
```

```
function [K] = build_kernel (X, f, f_param)
% Aceasta functie este utilizata pentru construirea matricii de kerneluri.
```

```
function [L] = cholesky (A)
% Aceasta functie implementeaza metoda Cholesky pentru descompunerea LU a unei
    matrici pozitiv semi-definite.
```

```
function [P] = get_lower_inverse (L)
% Aceasta functie inverseaza o matrice inferior triunghiulara (lower) folosind un
    algoritm de eliminare gaussiana.
```

```
function [a] = get_prediction_params (K, y, lambda)
% Functia primeste ca parametri matricea de kernel, valorile evaluate pentru
    inputurile pe care a fost construit K si parametrul lambda folosit pentru
    regularizare si trebuie sa intoarca vectorul coloana a ce contine estimari ai
    parametrilor folositi la evaluare (folositi metoda Cholesky).
```

```
function [x] = conjugate_gradient (A, b, x0, tol, max_iter)
% Functia implementeaza un algoritm pentru metoda gradientului conjugat, avand
    parametri A (o matrice pozitiv semi-definita), b (vectorul coloana rezultat al
    sistemului Ax = b), tol (toleranta maxima acceptata pentru ca o solutie sa fie
    valida), max_iter (numarul maxim de iteratii pentru rezolvarea sistemului).
```

```
function [a] = get_prediction_params_iterative (K, y, lambda)
% Acesta functie este utilizata pentru estimarea parametrilor folositi la evaluare.
% Functia primeste ca parametri matricea de kernel, valorile evaluate pentru
    inputurile pe care a fost construit K si parametrul lambda folosit pentru
    regularizare. Intoarce vectorul coloana a ce contine estimari ai parametrilor
    folositi la evaluare (folositi metoda gradientului conjugat).
```

```
function pred = eval_value(x, X, f, f_param, a)
% Aceasta functie este uilizata pentru calcularea estimarii produse de model.
% Pentru un vector oarecare x ca input (vector linie), X (toti vectorii folositi
    pentru construirea modelului), f (functia de kernel), f_param (al 3-lea parametru
    al functiei de kernel) si a (estimarea parametrilor folositi la evaluare) trebuie
    sa returnati predictia facuta pentru datele de intrare reprezentate de x.
```

3 Metoda 3: Generare de text

3.1 Introducere

Notă de început: Nu vă speriați, metoda nu este complicată, însă am explicat în acest document mai în detaliu despre lanțurile Markov și despre terminologia folosită în Învățarea Automată, terminologie cu care veți avea contact și în ani mai mari, poate chiar și în cariera voastră.

Cu toții sunteți familiari cu ChatGPT. GPT se traduce în *General Purpose Transformer*. Prin această metodă propusă, nu vom ajunge nici pe departe la nivelul lui ChatGPT, ci vom încerca însă să creăm un mini-AI bazat pe lanțuri Markov care poate genera text pe baza unui fișier de intrare (input).

3.2 Lanțuri Markov

Lanțurile Markov (sau Procesele Markov) sunt modele stocastice ce descriu o înșiruire de evenimente posibile, probabilitatea de producere a unui eveniment din lanț depinzând doar de probabilitea evenimentul anterior. Acestea sunt folosite în diverse tipuri de metode de prezicere (en: *forecasting*) cum ar fi procnozele meteo, și pot fi reprezentate grafic prin dependențe pe care voi le-ați reprezentat ca *grafuri orientate*.

În reprezentarea grafică, fiecare nod al grafului reprezintă o **stare** a unui sistem, și este legat cu celelalte noduri în funcție de probabilitatea de trecere de la starea respectivă la o stare adiacentă.

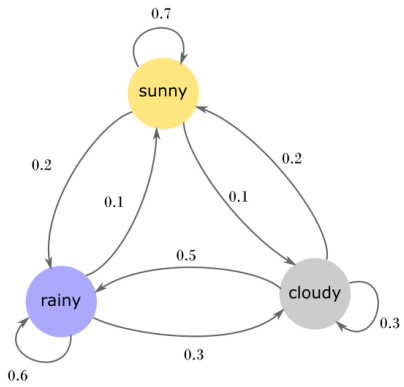


Figure 2: Exemplu de Lanț Markov pentru vreme.

În Figura 2 este reprezentat un lanț Markov ce caracterizează vremea de la o zi la alta. Din aceasta deducem că dacă într-o zi a plouat, atunci există o probabilitate de 0.6 să plouă și în următoarea zi, 0.1 să fie însorit, și 0.3 șanse să fie înnorat.

3.3 Reprezentarea unui text folosind lanțul Markov

Ne putem imagina un text ca o înșiruire de ”evenimente” (spuse sau scrise), unde fiecare eveniment (nod) este un cuvânt. Fie textul: **”Am mers ieri la piata si la magazin”**. Un lanț Markov pentru acesta arăta ca în Figura 3.

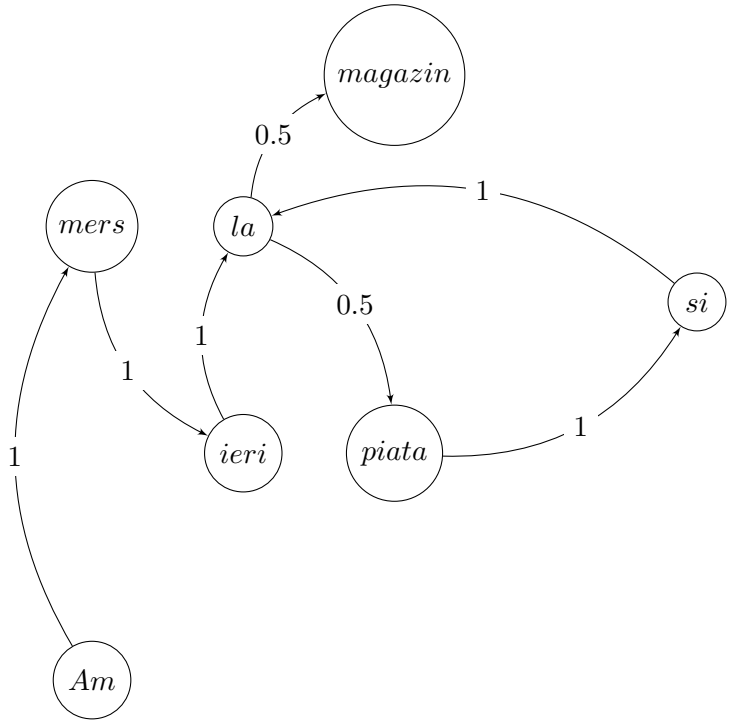


Figure 3: Lanț Markov pentru textul *"Am mers ieri la piata si la magazin"*.

După cum observați, deși **"la"** apare de 2 ori în text, el apare o singură dată în lanțul Markov, deoarece reprezintă o singură stare a sistemului. Acest lanț îl putem "citi" astfel:

- După "Am" urmează "mers";
- După "si" urmează "la";.
- După "la" urmează "piata" cu probabilitate 0.5 sau "magazin" cu probabilitate de 0.5.

3.4 Descrierea metodei

Setul de antrenament

Setul de antrenament (en: *Training set*) este setul de date pe care le folosim pentru antrenarea modelului. În cazul nostru, acest set este constituit din mai multe texte precum **Tiny Shakespeare**[3] care sunt folosite pentru a antrena modelul, adică pentru a construi lanțul Markov.

Token-uri

Token-urile reprezintă primul pas în procesarea datelor de intrare. În cazul nostru token-urile le obținem prin extragerea cuvintelor din text și eliminarea semnelor de punctuație.

În realitate, în cazul GPT, modul de selectare a token-urilor este unul mult mai complex, similar cu cel din limbajul natural. De exemplu, folosind token-ul "metod" cu unul dintre token-urile "a", "ele", "ic", creăm cuvintele "metoda", "metodele", "metodic".

[3] `https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt`

În exemplul anterior **"Am mers la piata si la magazin"**, token-urile sunt "Am", "mers", "la", "piata", "si", "la", "magazin" (în ordine). Datorită naturii secvențiale a task-ului pe care îl facem (e.g. într-o propoziție ordinea textului contează), vom păstra **toate** token-urile chiar dacă se repeta, **in ordine**.

"Am mers la piata si la magazin" → ["Am" "mers" "la" "piata" "si" "la" "magazin"].

Token-urile se vor extrage din **Training Set**, pentru a genera lanțul Markov care se va folosi pentru prezicerea urmatorului cuvant din **Testing Set**.

K Secvente / Sliding K Window

Lanțul Markov din Figura 3 este unul superficial, deoarece starea curentă depinde doar de ultimul token (cuvânt) din șir, adică starea anterioară. Un lanț Markov mai adânc are capacitatea de a prezice următorul cuvânt știind ultimele k stări. Spre exemplu un lanț Markov de adâncime $K = 2$ ar arăta astfel (în varianta textuală):

- ["Am", "mers"] → "la"
- ["mers", "la"] → "piata"
- ["la", "piata"] → "si"
- ["piata", "si"] → "la"
- ["si", "la"] → "magazin"

Practic am creat o "fereastră glisantă" (en: *sliding window*) de dimensiune 2. Am obținut 5 2-Secvente. Ideea din spatele acestui concept este oferirea de context cuvintelor. Un exemplu ar putea fi ["lanț", "Markov"] și ["lanț", "muntos"]. În acest caz, cuvântul "lanț" se referă la 2 lucruri diferite iar dacă am fi folosit $K = 1$, acest context s-ar fi pierdut.

Noi ne dorim o generare de text cât mai naturală (adică K ar trebui să fie cât mai mare), dar totuși vrem să îi dăm modelului și libertate de "creație" (asocieri de cuvinte din contexte diferite, adică K ar trebui să fie cât mai mic. Alegerea lui K se face în funcție de dimensiunea setului de antrenare. Cu cât este mai mare setul, cu atât alegem K mai mare. Training set-ul nostru este unul relativ mic, așa că vom folosi $K = 2$.

Features & Labels

În general, seturile de date sunt destul de complexe. Înainte de antrenarea propriu-zisă, un pas foarte important este extragerea din acestea a unor caracteristici. Aceste caracteristici se numesc "*features*". Practic, acest pas impune găsirea unui mod simplu și eficient de reprezentare a datelor.

Unui set de caracteristici îi asociem o etichetă, adică label. Acest label descrie ce reprezintă caracteristicile respective. O reprezentare foarte simplă a unui model este o funcție ce asociază unui set de caracteristici, un label:

$$f(X) = y.$$

În cazul nostru, feature-urile sunt K-Secventele extrase din **Corpul** de cuvinte (token-uri), iar label-urile sunt cuvintele următoare. Avem următoarele definții.

Training Set: set de date pentru care avem deja un label asociat fiecărui set de caracteristici. Modelul învață ce reprezintă fiecare set de caracteristici.

Testing Set: set de date pentru care vrem să prezicem label-ul asociat fiecărui set de caracteristici. Modelul este testat pentru a vedea cât de bine face acest lucru.

3.5 Detalii de implementare a metodei

Extragerea Feature-urilor și a Label-urilor

Pentru o secvență S de cuvinte, de lungime n cu elementele $S[1], S[2], ..., S[n]$, din training set, deducem următoarele feature-uri (k-secvențe) și label-uri.

- Features → Label
- S[1:k] → S[k+1]
- S[2:k+1] → S[k+2]
- S[3:k+2] → S[k+3]
- S[4:k+3] → S[k+4] ....

De exemplu, pentru token-urile [”Am”, ”mers”, ”la”], vom avea secvența (feature) [”Am”, ”mers”] și label-ul ”la”.

Codificarea și decodificarea feature-urilor și label-urilor

Deoarece calculatorului îi este mult mai ușor să lucreze cu vectori de numere decât cu cuvinte, vom atribui fiecărei K-secvențe (feature) și fiecărui label (cuvânt) un identificator numeric. Cum fiecare label face parte din corpul de cuvinte, o metodă simplă ar fi să creăm un set ordonat de cuvinte unice, și să atribuim fiecărui cuvânt index-ul din acel set. Spre exemplu:

”Am mers ieri la piata si la magazin”

Cuvintele, adică token-urile extrase vor fi:

Corpus-Tokens = [**”Am” ”mers” ”ieri” ”la” ”piata” ”si” ”la” ”magazin”**]

După ce ordonăm și eliminăm duplicatele,

Word_Set = [**”Am” ”ieri” ”la” ”magazin” ”mers” ”piata” ”si”**]

Folosindu-ne de acest set putem deduce codificările pentru fiecare dintre cuvinte, acestea fiind index-ul din set aferent cuvântului. De exemplu, $word_{index}(”Am”) = 1$, $word_{index}(”ieri”) = 2$, etc, unde $word_{index}$ reprezintă codificarea label-urilor. Procedăm la fel și pentru features. Amintiți-vă că feature-urile noastre sunt K-secvențele extrase din corpusul de cuvinte.

Corpus-Tokens = [**”Am” ”mers” ”ieri” ”la” ”piata” ”si” ”la” ”magazin”**]

Alegem $K = 2$ pentru exemplificare.

K-Secv = [**”Am mers” ”mers ieri” ”ieri la” ”la piata” ”piata si” ”si la” ”la magazin”**]

Ordonăm și eliminăm duplicatele.

K-Secv-Set = [**”Am mers” ”ieri la” ”la piata” ”la magazin” ”mers ieri” ”piata si” ”si la”**]

Fiecare secvență are astfel un index în acest set sortat. De exemplu $ksecv_{index}(”Am\ mers”) = 1$, $ksecv_{index}(”ieri\ la”) = 2$, etc. Am creat cele 2 dicționare: $word_{index}$ și $ksecv_{index}$ care codifică label-uri, respectiv feature-uri. Ca și exemplu, feature-ului 1 (”Am mers”) îi corespunde label-ul 2 (”ieri”), feature-ului 4 (”la piata”) îi corespunde labelul 7 (”si”), șamd.

Observați că acele K-secvențe sunt un șir de caractere compus din cuvintele selectate, deoarece Octave nu permite dicționare cu alte tipuri de cheie în afara de string sau number. În cazul nostru ne-ar fi trebuit ca și cheie un vector de cuvinte.

3.6 Lanțul Markov ca matrice stocastică

Folosindu-ne de aceste codificări (word_index și ksecv_index), putem memora lanțul Markov asociat sub forma unei matrice. Punem pe poziția (i, j) numărul de apariții ale cuvântului cu index j după K-secvența cu index i.

Spre exemplu, considerând setul de antrenament **"a b a b c"**:

- Textul se token-izează și apar următoarele label-uri: **"a", "b", "a", "b", "c"**;
- Label-urile ordonate și unice devin **"a", "b", "c"** cu index 1, 2, respectiv 3;
- Pentru $K = 2$ rezultă urmatoarele secvențe: **"a b", "b a", "a b", "b c"**;
- Secvențele ordonate și unice devin **"a b", "b a", "b c"** cu index 1, 2 respectiv 3.

Prin urmare, după feature-ul "a b" apare "a" o dată sau "c" o dată, deci matricea ce descrie lanțul Markov va avea 1 pe poziția (1,1) și (1,3). Matricea va arăta astfel:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

După cum observați, fiecare feature poate avea mai multe label-uri posibile cu probabilități diferite. Dacă împart fiecare element din linie la suma elementelor de pe linia respectivă obținem matricea stocastică pe linii:

$$\begin{bmatrix} 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Pe poziția (i, j) avem acum probabilitatea ca label-ul (sau cuvântul) cu index j să urmeze după feature-ul (K-secvența) j. Cu o astfel de matrice putem prezice următorul cuvânt dintr-un text:

- Extragem ultimele K cuvinte din text pentru a construi secvența;
- Ne uităm în dicționarul de codificări și codificăm k-secvența formată (obținem un index i);
- Ne uităm pe linia i din matricea stocastică, iar mai apoi folosim un generator de numere numit **weighted random number generator**. Acesta ne va genera un număr de la 1 la $size(word_{set})$ cu greutatea specificată de linia selectată.

De exemplu, dacă se alege linia 1, adică $[0.5, 0, 0.5]$, generatorul va genera 1 în 50% dintre cazuri, 3 in 50% din cazuri și niciodată 2.

3.7 Funcțiile din cadrul bibliotecii

Cerința constă în implementarea unor funcții pentru funcționalitățile de mai sus. În biblioteca de funcții aveți fișierele "data/**.txt" și "test/test.txt". "data**.txt" conține seturile de antrenare, adică textul pe baza căruia se va construi matricea stocastică.

```
function tokens = split_input (filePath)
% Aceasta functie este utilizata pentru tokenizarea fisierului de intrare. Preia
    calea fisierului si intoarce token-urile. In cazul nostru, ne dorim si
    tokenizarea semnelor de punctuatie. Spre exemplu, textul "Salut, frate!", va avea
    4 token-uri: "Salut", ",", "frate", "!", nu doar doua. Pentru a verifica
    consistent punctuatia, va veti implementa o functie care verifica daca un
    caracter este sau nu semn de punctuatie, pe care o puteti folosi.
% Hint: textscan, care intoarce un cell-array.
```

```
function is_punc = punctuation (input)
% Aceasta functie este utilizata pentru a testa daca un caracter este semn de
    punctuatie: ',' '.' '?' ';' ':' backslash '(' ')' '!' '"' "'" '-'.
```

```
function retval = distinct_words (tokens)
% Aceasta functie intoarce tokenurile sortate si unice.
```

```
function B = k_secv (A, k)
% Aceasta functie returneaza un cell-array de k-secvente pentru un cell-array
    unidimensional. E.g.: Pt [a, b, c, a, d], returneaza [abc, bca, cad] daca k=3.
```

```
function unique_cell_array = distinct_k_secv (cell_array)
% Aceasta functie intoarce k-secventele sortate si unice.
```

```
function retval = word_idx (distinct_wds)
% Aceasta functie intoarce un dictionar care contine indecsii asociati fiecarui label.
```

```
function retval = k_secv_idx (distinct_k_sec)
% Aceasta functie intoarce un dictionar care contine indecsii asociati fiecarui
    feature (k-secventa). // Hint1: Aceste 2 functii vor fi similare, însa este de
    preferat sa fie 2 entitati diferite pentru a va obisnui cu ideea ca uneori
    label-urile si feature-urile nu fac parte din acceasi familie. În cazul nostru,
    ambele sunt texte, însa în unele aplicatii pot fi imagini sau audio. // Hint2:
    containers.Map
```

```
function retval = stochastic_matrix (k_secv_corpus, corpus_words, words_idx,
    k_secv_idx, k)
% Aceasta functie creeaza matricea stochastica, avand la dispozitie cele 2 dictionare
    de codificari, respectiv k-secventele în ordinea în care au fost date în
    "input.txt", respectiv token-urile (corpus_words). K-secventei care incepe la
    pozitia j ii va corespunde cuvantul care este la pozitia j+k in input.txt.
```

```
function probs = sample_next_word (text, words_idx, k_secv_idx, k, stoch)
% Aceasta functie intoarce probabilitatile (linia din matrice) aferente ultimei
    k-secvente din text.
```

```
function retval = sample_n_words (text, word_set, kscv_set, k, stoch, word_set, n)
% Aceasta functie genereaza n cuvinte pornind de la textul dat astfel: Genereaza un
    cuvant, il appenduieste la text, si apoi genereaza alt cuvant pentru k-secvente
    ultima, pe care iar il adauga la final etc.
```

Puteți testa codul folosind scriptul **"tema1_script.m"** care va secvența aceste funcții pentru a genera 10 cuvinte consecutive pornind de la conținutul fișierului "test.txt". Acest script nu este rulat de către checker. Puteți pune orice alt fișier text vă doriți.

4 Regulament

Regulamentul general al temelor se găsește pe platforma Moodle[4]. Vă rugăm să îl citiți integral înainte de continua cu regulile specifice acestei teme.

4.1 Arhiva temei de casă

Soluția temei se va trimite ca o arhivă **zip**. Numele arhivei trebuie să fie de forma **Grupă_-NumePrenume_TemaX.zip** - de exemplu: `313CA_Adrian-NicolaeAriton_Tema1.zip`.

Arhiva temei trebuie să conțină doar cele 3 directoare conținând fișierele sursă (`anomaly_detection`, `kernel_regression` și `stochastic_text_generation`) împreună cu fișierul README/README.md.

Numele şi extensiile fișierelor auxiliare create de voi **NU** trebuie să conțină spații sau majuscule, cu excepția fișierului README (care este un nume scris cu majuscule și poate avea extensia *md*, în cazul în care optați pentru formatul Markdown).

Nerespectarea oricărei reguli din această secțiune aduce un punctaj **NUL** pe temă.

4.2 Punctaj

Distribuirea punctajului:

- Anomaly Detection: 20 p
- Kernel Regression: 35 p
- Stochastic Text Generation: 45 p
- Modularizare, claritatea codului și explicațiile din README(.md): 10p

ATENȚIE! Punctajul maxim pe temă este 100p. Acesta reprezintă 1p din nota finală la această materie.

4.3 Reguli și precizări

- Punctajul pe teste este cel acordat de scriptul de *checker.py*, rulat pe VMChecker. Echipa de corectare își rezervă dreptul de a depuncta pentru orice încercare de a trece testele fraudulos (de exemplu prin hardcodare).
- Punctajul pe calitatea explicațiilor și a codului se acordă în mai multe etape:
  - Codul sursă trebuie să fie însoțit de un fișier README(.md) care trebuie șă conțină informațiile utile pentru înțelegerea funcționalității, modului de implementare și utilizare a soluțiilor cerute. Acesta evaluează, de asemenea, abilitatea voastră de a documenta complet și concis programele pe care le produceți și va fi evaluat de către echipa de asistenți. In funcție de calitatea documentației, se vor aplica depunctări sau bonusuri.
  - Deprinderea de a scrie cod sursă de calitate, este un obiectiv important al materiei. Sursele greu de înțeles, modularizate neadecvat sau care prezintă hardcodări care pot afecta semnificativ mentenabilitatea programului cerut, pot fi depunctate adițional.

[4] `https://curs.upb.ro/2023/pluginfile.php/265759/mod_resource/content/0/Regulament_MN.pdf`

  - În această etapă se pot aplica depunctări mai mari de 30p.
  - Deși nu impunem un anumit standard de coding style, ne așteptăm să întâlnim cod lizibil, documentat corespunzător. Erorile grave de coding style (cod ilizibil, variabile denumite nesugestiv, hardcodarea sau lipsa comentariilor de orice fel) vor fi depunctate corespunzător.

Alte precizări

- Implementarea se va face in limbajul **GNU Octave**, iar tema testată **DOAR** într-un mediu **LINUX**. Nerespectarea acestor reguli aduce un punctaj **NUL**.
- Tema trebuie trimisă sub forma unei arhive pe site-ul cursului curs.upb.ro și pe VMChecker, în secțiunea dedicată materiei [5]
- Tema poate fi submisă de oricâte ori fără depunctări până la deadline. Mai multe detalii se găsesc în regulamentul de pe ocw.
- Ultima temă submisă pe vmchecker poate fi rulată de către responsabili de mai multe ori în vederea verificării faptului că nu aveți buguri în codul sursă. Vă recomandăm să verificați **local** tema de mai multe ori pentru a verifica că punctajul este mereu același, apoi să încărcați tema.
- Temele vor fi testate anti-plagiat. Este interzisă publicarea pe forum și în orice spațiu public (GitHub) a întregului cod sau a unor porțiuni din cod care reprezintă soluții la task-urile propuse. Nu este permisă colaborarea de orice fel în vederea realizării temelor de casă. Soluțiile care nu respectă aceste criterii vor fi punctate cu 0 (zero) puncte.
- Preluarea de cod din resurse publice este permisă doar în contextul menționării sursei în comentarii **și** în README, cu excepția resurselor care reprezintă rezolvări directe ale task-urilor din temă. Soluțiile care nu respectă aceste criterii vor fi punctate cu 0 (zero) puncte.
- Este interzisă folosirea ChatGPT sau a oricărei formă de LLM în rezolvarea temei de casă. Soluțiile care nu respectă acest criteriu vor fi punctate cu 0 (zero) puncte.

[5] `https://vmchecker.cs.pub.ro/ui/#MN`