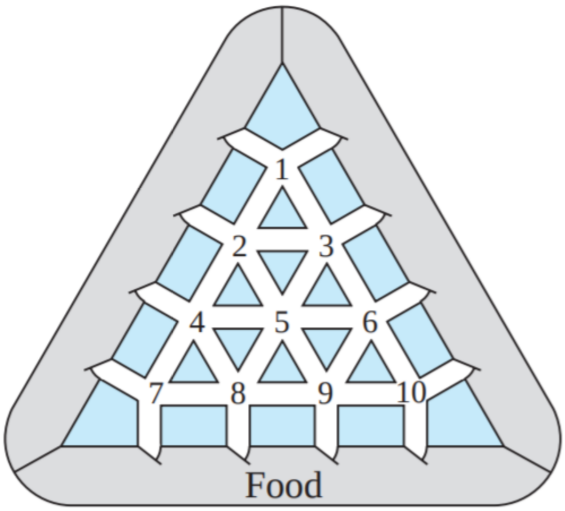

Figura 1: Diagramă a labirintului.

1 Sparse Jacobi (part-1) - 40p

1.1 Scop

Scopul acestei părți este alcătuit din următoarele:

- familiarizarea cu GNU Octave;
- dobândirea capacității de a lucra cu matrici rare în forme comprimate;
- rezolvarea unui sistem liniar $Ax = b$ printr-o metodă iterativă.

1.2 Enunț

În Figura 1 este o diagramă a unui labirint folosit într-un experiment de laborator. Experimentul începe prin a pune un șoarece la una dintre cele 10 intersecții interioare ale labirintului. După ce șoarecele iese în coridorul exterior (gri), acesta nu se mai poate întoarce în labirint. Când șoarecele este într-o intersecție interioară, calea pe care alege să meargă este aleatorie. Care este probabilitatea ca șoarecele să ajungă la mâncare (coridorul gri de jos) când începe în intersecția i?

Fie probabilitatea câștigului (a ajunge la mâncare) începand din intersecția i reprezentată de p_i. Putem forma o ecuație liniară folosind p_i și probabilitățile asociate cu intersecțiile vecine intersecției i. De exemplu, în prima intersecție șoarecele are o probabilitate de $\frac{1}{4}$ să meargă în dreapta-sus și să piardă, o probabilitate de $\frac{1}{4}$ să meargă în stânga-sus și să piardă, o probabilitate de $\frac{1}{4}$ să meargă în dreapta-jos (unde are probabilitatea p_3 de a câștiga) și o probabilitate de $\frac{1}{4}$ să meargă în stânga-jos (unde are probabilitatea p_2 de a câștiga). Rezultă următoarea ecuație:

$$p_1 = \frac{1}{4}(0) + \frac{1}{4}(0) + \frac{1}{4}p_2 + \frac{1}{4}p_3$$

.
În mod analog, celelalte nouă probabilități pot fi reprezentate de următorul sistem de ecuații.

$$p_2 = \frac{1}{5}(0) + \frac{1}{5}p_1 + \frac{1}{5}p_3 + \frac{1}{5}p_4 + \frac{1}{5}p_5$$
$$p_3 = \frac{1}{5}(0) + \frac{1}{5}p_1 + \frac{1}{5}p_2 + \frac{1}{5}p_5 + \frac{1}{5}p_6$$

$$p_4 = \frac{1}{5}(0) + \frac{1}{5}p_2 + \frac{1}{5}p_5 + \frac{1}{5}p_7 + \frac{1}{5}p_8$$
$$p_5 = \frac{1}{6}p_2 + \frac{1}{6}p_3 + \frac{1}{6}p_4 + \frac{1}{6}p_6 + \frac{1}{6}p_8 + \frac{1}{6}p_9$$
$$p_6 = \frac{1}{5}(0) + \frac{1}{5}p_3 + \frac{1}{5}p_5 + \frac{1}{5}p_9 + \frac{1}{5}p_10$$
$$p_7 = \frac{1}{4}(0) + \frac{1}{4}(1) + \frac{1}{4}p_4 + \frac{1}{4}p_8$$
$$p_8 = \frac{1}{5}(1) + \frac{1}{5}p_4 + \frac{1}{5}p_5 + \frac{1}{5}p_7 + \frac{1}{5}p_9$$
$$p_9 = \frac{1}{5}(1) + \frac{1}{5}p_5 + \frac{1}{5}p_6 + \frac{1}{5}p_8 + \frac{1}{5}p_{10}$$
$$p_{10} = \frac{1}{4}(0) + \frac{1}{4}(1) + \frac{1}{4}p_6 + \frac{1}{4}p_9$$

Rescrierea acestor ecuații în forma standard produce următorul sistem de 10 ecuații liniare cu 10 necunoscute.

$$4p_1 - p_2 - p_3 = 0$$
$$-p_1 + 5p_2 - p_3 - p_4 - p_5 = 0$$
$$-p_1 - p_2 + 5p_3 - p_5 - p_6 = 0$$
$$-p2 + 5p_4 - p_5 - p_7 - p_8 = 0$$
$$-p_2 - p_3 - p_4 + 6p_5 - p_6 - p_8 - p_9 = 0$$
$$-p_3 - p_5 + 5p_6 - p_9 - p_10 = 0$$
$$-p_4 + 4p_7 - p_8 = 1$$
$$-p_4 - p_5 - p_7 + 5p_8 - p_9 = 1$$
$$-p_5 - p_6 - p_8 + 5p_9 - p_10 = 1$$
$$-p_6 - p_9 + 4p_{10} = 1$$

Sistemul se poate scrie în forma matriceală, după cum se poate vedea mai jos.

$$A = \begin{bmatrix} 4 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 5 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 5 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 5 & -1 & 0 & -1 & -1 & 0 & 0 \\ 0 & -1 & -1 & -1 & 6 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 5 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & -1 & 5 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & -1 & 5 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Folosind metoda iterativă **Jacobi** cu o aproximare inițială $p_1 = p_2 = ... = p_{10} = 0$ găsim după câteva iterații o aproximare:

$$p_1 = 0.090, p_2 = 0.180$$
$$p_3 = 0.180, p_4 = 0.298$$
$$p_5 = 0.333, p_6 = 0.298$$
$$p_7 = 0.455, p_7 = 0.522$$
$$p_9 = 0.522, p_{10} = 0.455$$

Deoarece atât matricea A, cât și matricea de iterație G_J folosită pentru metoda Jacobi pot fi matrici rare, vrem să reținem matricea de iterație în forma **CSR** (Compressed Sparse Row Format).

Aceasta constă în:

- **n** - numărul de linii din matrice;
- **nz** - numărul de valori nenule din matrice;
- un vector **values** de lungime **nz** care conține toate valorile nenule din matrice obținute rând cu rând;
- un vector **colind** de lungime **nz** care conține toți indicii coloanelor elementelor din matrice stocate în **values**;
- un vector **rowptr** de lungime **n + 1** care conține pe primele **n** poziții indicii (poziția din vectorii **values** sau **colind**) primelor elemente de pe fiecare rând; pe ultima poziție se pune valoarea **nz + 1**.

Matricea

$$A = \begin{bmatrix} 100 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

stocată în forma **CSR** este:

- `values = [100 2 3 4 5 6 7 8 9 10 11 12];`
- `colind = [1 4 1 2 4 1 3 4 5 3 4 5];`
- `rowptr = [1 3 6 10 12 13];`

Pentru a înmulți o matrice A în forma **CSR** cu un vector x astfel încât $y = Ax$ se poate folosi algoritmul următor (pe care îl veți găsi în scheletul de cod):

```
function [y] = csr_multiplication(values, colind, rowptr, x)
  n = length(x);
  y = zeros(n, 1);
  for i = 1:n
    for j = rowptr(i):rowptr(i+1) - 1
      y(i) = y(i) + values(j) * x(colind(j));
    end
  end
endfunction
```

1.3 Cerința

Studentul trebuie să implementeze următoarele funcții:

- `function [A, b] = generate_probabilities_system(rows)`
  - Această funcție primește ca parametru numărul de rânduri $rows > 2$ pe care le are diagrama din Figura 1 și returnează sistemul de ecuații reprezentat de matricea A (în forma sa densă) și de vectorul b.
- `function [values, colind, rowptr] = matrix_to_csr(A)`
  - Această funcție primește ca parametru o matrice A (în forma sa densă) și returnează vectorii ce reprezintă forma sa **CSR**.
- `function [G_J, c_J] = Jacobi_factorization(A, b)`
  - Această funcție primește ca parametri o matrice A (în forma sa densă) și un vector b ce reprezintă un sistem liniar de ecuații și returnează matricea de iterație G_J și vectorul de iterație c_J reprezentative pentru metoda **Jacobi**.
- `function [x] = Jacobi_sparse(G_values, G_colind, G_rowptr, c, tol)`
  - Această funcție primește ca parametri matricea de iterație G_J în forma sa **CSR**, vectorul de iterație c_J și o toleranta *tol* sub care se pot opri iterațiile și returnează vectorul x ce reprezintă soluția sistemului;
  - Vectorul inițial cu care se aproximează soluția este egal cu 0.

Studentul primește deja implementată funcția **csr_multiplication.m**, dar trebuie să o includă în arhiva trimisă pentru evaluare!

1.4 Punctaj și evaluare

Toate funcțiile ce trebuie implementate au punctaj egal. Evaluarea se poate face folosind fișierul **checker/checker-part-1/checker_part_1.m** după ce s-au copiat toate sursele necesare în **checker/checker-part-1/**.

2 K-Means (part-2) - 30p

2.1 Scop

Scopul acestei părți este alcătuit din următoarele:

- utilizarea funcționalităților limbajului GNU Octave în mod eficient;
- implementarea unui algoritm n-dimensional de clustering;
- descoperirea elementelor de bază ale Învățării Automate Nesupervizate (Unsupervised ML).

2.2 Enunț

2.2.1 Ce înseamnă clustering?

Algoritmii de clustering sunt folosiți pentru a grupa anumite obiecte în mulțimi separate astfel încât fiecare obiect să aibă mai multe caracteristici în comun cu obiectele din mulțimea căreia îi aparține, decât cu alte obiecte. În cadrul acestei părți, se cere implementarea unui astfel de algoritm denumit K-Means.

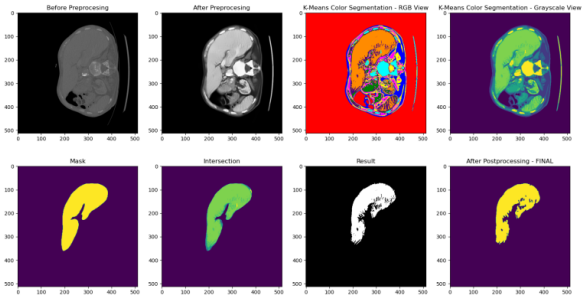

Figura 2: Selecția unei anumite componente dintr-o imagine

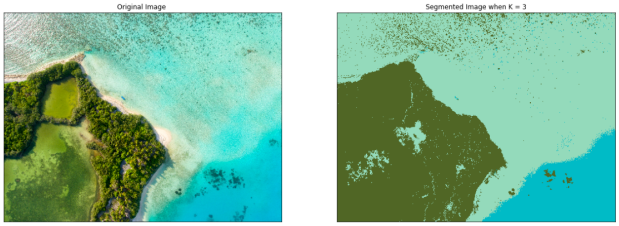

Figura 3: Segmentarea unei imagini după culoare

Figura 4: Gruparea pe categorii de cumpărători în funcție de salariu - prețul cumpărăturilor

2.2.2 De ce clustering?

Algoritmii de clustering se pot întâlni într-o multitudine de aplicații, precum image processing (Figura 2 sau Figura 3), segmentarea tipurilor de clienți pe grupuri (Figura 5) sau doar găsirea unor grupuri de elemente cu caracteristici asemănătoare. De obicei, datele pe care dorim să le prelucrăm trebuie proiectate într-un spațiu favorabil algoritmului K-means (trebuie codificate numeric).

Inițializarea algoritmului se poate face în multe moduri, însă pentru a putea testa uniform, vom impune un tip de inițializare descris mai jos.

Algoritmul K-Means pentru o mulțime de puncte **points** și **NC** clustere:

1. Inițializare
   - Se creează NC clustere (liste de puncte), unde cluster-ul i conține toate punctele $mod\ NC = i$ (punctul i, punctul $i + NC$ etc.);
   - Se creează NC centroizi, fiecare fiind media (centrul de masă) al unuia dintre clusterele create anterior; centroidul i va fi media punctelor clusterului i.
2. Se inițializează NC grupuri (liste) goale de puncte, fiecare având câte un reprezentant, anume unul dintre cei NC centroizi;
3. Fiecare punct din **points** este atribuit grupului (listei) reprezentat de cel mai apropiat centroid folosind distanța euclidiană;
4. Recalculăm pozițiile centroizilor ca fiind mediile (centrele de masă) punctelor atribuite fiecarui grup în parte;
5. Repetăm pașii 2, 3 și 4 până când pozițiile centorizilor nu se mai modifică.

2.3 Cerința

Studentul trebuie să implementeze următoarele funcții:

- `function [centroids] = clustering_pc(points, NC)`
  - Această funcție primește ca parametri un vector de puncte D-dimensionale (matrice $N \times D$) *points* și numărul de clustere în care trebuie segmentate punctele NC și trebuie să returneze un vector de puncte *centroids* ce reprezintă mediile (centrele de masa) ale clusterelor.
- `function [cost] = compute_cost_pc(points, centroids)`
  - Această funcție primește ca parametri un vector de puncte D-dimensionale (matrice $N \times D$) *points* și un vector de centroizi *centroids* și returnează **costul** clustering-ului, adică suma distanțelor euclidiene de la fiecare punct din *points* la centroidul clusterului căruia îi aparține. Acest cost trebuie să fie minim.

2.4 Punctaj

Toate funcțiile ce trebuie implementate au punctaj egal. Evaluarea se poate face folosind fișierul **checker/checker-part-2/checker_part_2.m** după ce s-au copiat toate sursele necesare în **checker/checker-part-2/**.

2.5 Optional: Vizualizarea datelor D-dimensionale

După cum puteți intui, nu se pot reprezenta vizual cu ușurință puncte cu mai mult de 3 dimensiuni. Cu toate acestea, există metode de reducere a dimensionalității care ne ajută să integram date D-dimensionale într-un spațiu pe care îl putem vizualiza. O astfel de tehnică este **t-SNE** (t-distributed stochastic neighbour embedding). Acest algoritm nu face parte din programa de Metode Numerice, dar puteți utiliza codul inclus

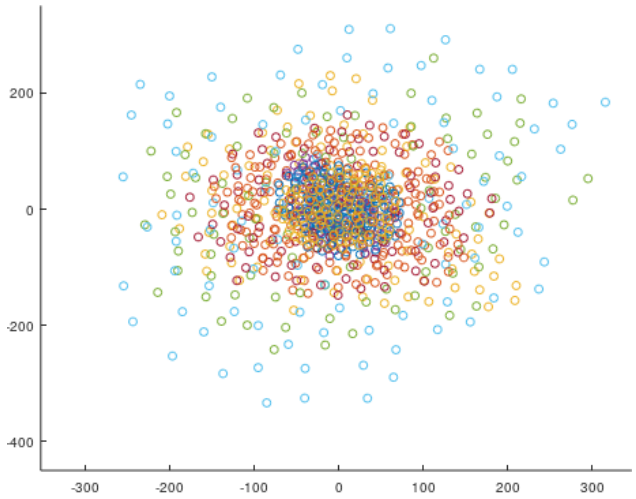

Figura 5: Clusterig pe 100 de puncte 20-D

în arhiva găsită aici pentru a vizualiza puncte D-dimensionale (implicit clustering-ul lor) într-un spatiu 2D. Codul se poate utiliza astfel:

Algorithm 1: t-SNE

$data = tsne(X)$;
$scatter(data(:, 1),\ data(:, 2))$;

3 Householder prediction (part-3) - 30p

3.1 Scop

Scopul acestei părți este alcătuit din următoarele:

- utilizarea funcționalităților limbajului GNU Octave în mod eficient;
- rezolvarea unui sistem liniar $Ax = b$ printr-o metodă de factorizare ortogonală;
- descoperirea elementelor de bază ale Învățării Automate Supervizate (Supervised ML).

3.2 Enunț

3.2.1 Histograma RGB

Imaginile pot fi reprezentate în spațiul RGB al culorilor, în care fiecare pixel este definit de un triplet (r, g, b) care reprezintă intensitatea roșului, a verdelui și, respectiv, a albastrului. Un mod prin care poate fi reprezentată distribuția culorilor într-o imagine este prin analizarea unei histograme, precum cea din Figura 6. O histogramă măsoară frecvența de apariție a pixelilor care au anumite valori pentru r, g sau b. Se observă că există câte o histogramă separată pentru fiecare dimesniune a spațiului culorilor (respectiv pentru roșu, verde și albastru).

Un model folosit frecvent este cel în care valorile lui r, g, b sunt întregi din intervalul [0, 255]. O histograma pentru R, de exemplu, poate avea 256 de valori pe axa orizontală, caz în care vom obține pe a i-a verticală numărul de pixeli din componența R care au valoarea i. Dacă pe axa orizontală există **count_bins** valori, atunci pe a i-a verticală se va regăsi numărul de pixeli care au valoarea componentei R în intervalul $[i\frac{256}{count\_bins}, i\frac{256}{count\_bins} + \frac{256}{count\_bins})$. În mod analog se procedează pentru componentele G și B ale imaginii.

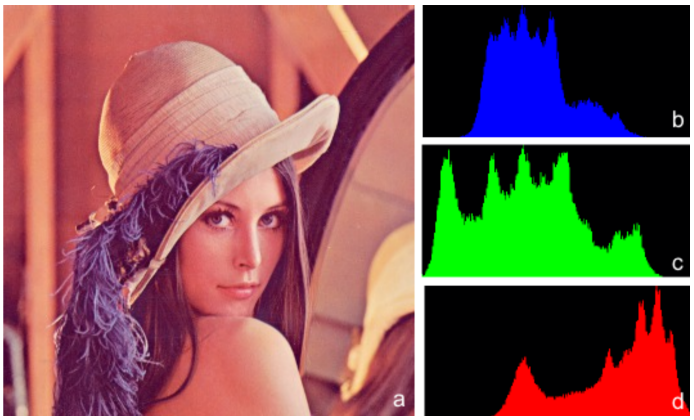

Figura 6: Histograma RGB. (a) Imaginea 250x250 (b) Canalul albastru (c) Canalul verde (d) Canalul roșu.

3.2.2 Histograma HSV

Un spațiu al culorilor mai puțin familiar decât RGB este HSV (Hue, Saturation, Value). La fel ca la RGB, fiecare pixel poate fi reprezentat ca un triplet (h, s, v) unde fiecare valoare din triplet îi corespunde unei dimensiuni din spațiul de reprezentare HSV. Există metode prin care se poate face conversia între RGB și HSV. Cum imaginile pe care le aveți la dispoziție sunt reprezentate în formatul RGB, ne interesează conversia RGB2HSV. Puteți folosi algoritmul de mai jos pentru a realiza conversia unui singur pixel din spațiul RGB în spațiul HSV. Țineți cont de faptul că, spre deosebire de RGB, unde fiecare componentă lua valori între 0 și 255, la HSV domeniile dimensiunilor sunt diferite: H ia valori intre 0 și 360, în timp ce S și V iau valori de la 0 la 100. De aceea, pentru ușurință, e util (dar nu obligatoriu) să normăm valorile lui H, S și V, pentru a le aduce în intervalul $[0, 1]$.

Atenție! Modurile în care se face operația de binning pentru intervale diferite ($[0, 1]$ vs. $[0, 100]$) pot să difere.

Algorithm 2: RGB2HSV

$R' \leftarrow R/255$;
$G' \leftarrow G/255$;
$B' \leftarrow B/255$;
$C_{max} \leftarrow max(R', G', B')$;
$C_{min} \leftarrow min(R', G', B')$;
$\Delta \leftarrow C_{max} - C_{min}$;
if $\Delta = 0$ **then**
  $H = 0$;
else
  if $C_{max} = R'$ **then**
    $H \leftarrow 60°(\frac{G'-B'}{\Delta} mod 6)$;
  if $C_{max} = G'$ **then**
    $H \leftarrow 60°(\frac{B'-R'}{\Delta} + 2)$;
  if $C_{max} = B'$ **then**
    $H \leftarrow 60°(\frac{R'-G'}{\Delta} + 4)$;
$H \leftarrow \frac{H}{360°}$;
if $C_{max} = 0$ **then**
  $S = 0$;
else
  $S = \frac{\Delta}{C_{max}}$;
$V \leftarrow C_{max}$;

3.2.3 Clasificare poze

Pentru a rezolva problema de clasificare, ne vom folosi de histogramele (RGB sau HSV) imaginilor din setul de date. Vom folosi un model matematic pentru a surprinde distribuția datelor și pentru a determina suprafața care delimitează clasa pozelor cu pisici. Pentru a folosi acest model, e nevoie ca fiecare imagine să fie reprezentată printr-un vector de caracteristici (**feature vector**). În cazul nostru, vom considera că acest vector este chiar rezultatul întors de funcțiile care construiesc histogramele. Astfel, vom avea $count\_bins \cdot 3$ caracteristici pentru fiecare imagine. Putem construi o matrice X care conține pe fiecare linie vectorul cu caracteristici ale unei imagini. Astfel, avem $X \in^{N*M}$, unde N este numărul total de imagini din setul de date și M este numărul de feature-uri ale unei imagini ($M = count\_bins \cdot 3$).

De asemenea, fiecare imagine va avea asociata o etichetă care spune din ce clasă face parte aceasta. În cazul nostru este vorba doar de două clase: poze cu pisici sau poze fără pisici. Considerăm următoarea codificare: dacă imaginea cu numărul de ordine i este o poză cu pisici, atunci $y_i = 1$ și $y_i = -1$ altfel. Astfel, obținem un vector y cu N elemente care va conține label-urile imaginilor din setul de date.

Modelul pe care îl vom folosi este descris de un vector de parametri notat cu w, care are lungimea egala cu numărul de caracteristici ale fiecarei poze din setul de date plus 1. Astfel lungimea lui w va fi egala cu $count\_bins \cdot 3 + 1$. Vectorul w descrie modelul imaginilor în raport cu caracteristicile pe care le-ați ales voi. Plecând de la setul de date pe care îl aveți la dispoziție (numit și set de învățare/antrenare), trebuie să **învățați** valoarea vectorului w pe care îl veți folosi apoi ca să **preziceți** clasa în care se află o nouă imagine. Pentru a putea face predicții bune, este necesar ca w să descrie bine nu doar imaginile din setul de învățare, ci trebuie să poată generaliza în așa fel încât împărțirea în clase pe care o modeleaza să rămână valabilă și pentru imagini noi, care nu au fost luate în considerare la învățarea lui w. Dupa ce w este învățat, produsul scalar $y = w^T \cdot x$ (unde x este vectorul de caracteristici ale unei imagini) va determina clasa în care se află imaginea dată: dacă $y \geq 0$ atunci este o poză cu o pisică, altfel nu.

Pentru învățarea lui w trebuie rezolvat sistemul de ecuații $\widetilde{X}w = y$. $\widetilde{X}$ este matricea cu vectorii cu caracteristici X descrisă mai sus, la care se adaugă la final o coloană cu 1-uri (termeni numiti **bias**). Acest sistem de ecuații se va rezolva folosind metoda de factorizare QR, **Householder**, împreuna cu rezolvarea unui sistem superior triunghiular. În acest fel, se evită calcularea inversei lui $\widetilde{X}$.

3.3 Cerința

Studentul trebuie să implementeze următoarele funcții:

- `function [sol] = rgbHistogram(path_to_image, count_bins)`
  - Această funcție primește ca parametri calea către o imagine *path_to_image* și un număr de valori pentru axa orizontală a histogramei *count* și returnează un vector linie *sol* de lungime *count*.3 care reprezintă histograma RGB a pozei de la calea primită ca parametru;
  - HINT: se recomandă folosirea uneia dintre funcțiile **accumarray**, **histc**.
- `function [sol] = hsvHistogram(path_to_image, count_bins)`
  - Această funcție primește ca parametri calea către o imagine *path_to_image* și un număr de valori pentru axa orizontală a histogramei *count* și returnează un vector linie *sol* de lungime *count*.3 care reprezintă histograma HSV a pozei de la calea primită ca parametru;
  - HINT: se recomandă modularizarea codului folosind o funcție separată care transformă o poză din format RGB în format HSV și o funcție separată care realizează histograma cerută.
  - Este **strict interzisă** folosirea funcției implicite **rgb2hsv** sau a vreunei funcții asemănătoare; studentul trebuie să implementeze transformarea din **RGB** în **HSV**.
- `function [Q, R] = Householder(A)`

- Această funcție primește ca parametru o matrice A (nu neaparat pătratică) și returnează o matrice ortogonală Q și o matrice superior triunghiulară R obținute folosind factorizarea **Householder**.

- `function [x] = SST(A, b)`
  - Această funcție primește ca parametru o matrice A (nu neapărat pătratică) superior triunghiulară și un vector coloană b și returnează un vector coloană x care este soluția sistemului $Ax = b$.

- `function [X, y] = preprocess(path_to_dataset, histogram, count_bins)`
  - Această funcție primește ca parametri calea către un set de imagini *path_to_dataset*, tipul histogramei ("RGB" sau "HSV") și un număr de valori pentru axa orizontală a histogramei $count_bins$ și returnează o matrice de caracteristici X (nu $\widetilde{X}$) descrisă mai sus și un vector coloană de etichete y;
  - HINT: se recomandă inițializarea lui X cu dimensiunile sale exacte în loc de adăugarea iterativă a unor linii noi în X;
  - HINT: y se completează cu 1 pentru imagini cu pisici și cu -1 pentru imagini ce nu conțin pisici; numele directoarelor sunt sugestive.

- `function [w] = learn(X, y)`
  - Această funcție primește ca parametri o matrice de caracteristici X și un vector coloană de etichete y și returnează vectorul de parametri ai modelului w aflat cu ajutorul **Householder** și **SST**.

- `function [percentage] = evaluate(path_to_testset, w, histogram, count_bins)`
  - Această funcție primește ca parametri calea către un set de imagini de testare *path_to_testset*, vectorul de parametri ai modelului w, tipul de histogramă evaluată ("RGB" sau "HSV") și un număr de valori pentru axa orizontală a histogramei *count_bins* și returnează un procentaj real între 0 și 100 al numărului de imagini clasificate corect *percentage*.

Se dorește o acuratețe de predicție mai mare de **0.68**.

3.4 Punctaj

Evaluarea se poate face folosind fișierul **checker/checker-part-3/checker_part_3.m** după ce s-au copiat toate sursele necesare în **checker/checker-part-3/**.

4 Gradient Descent prediction (part-4) - 20p

4.1 Scop

Scopul acestei părți este alcătuit din următoarele:

- utilizarea unei metode iterative (Gradient Descent) pentru rezolvarea unui sistem liniar $Ax = b$, astfel continuând inițierea în Învățarea Supervizată (ML).

4.2 Enunț

Deoarece în cele mai multe cazuri numărul de poze dintr-un set de date este foarte mare, matricea de caracteristici X este rău condiționată ($cond(X) > 1e15$) și rezolvarea directă sau printr-o factorizare a sistemului $\widetilde{X}w = y$ introduce erori mari de aproximare.

În acest caz se folosește o metodă numită **Gradient Descent** care, plecând de la o aproximare aleatorie a vectorului de parametri w, aproximează succesiv noi valori pentru elementele lui w.

Rezolvarea ecuației $\widetilde{X}w = y$ găsește minimul global al funcției de cost $J = \frac{1}{2N}\sum_{i=1}^{N}(\widetilde{X}(i,:)\cdot w - y(i))^2$, unde N este numărul total de imagini din setul de date. Găsirea unui minim local (dar suficient de bun) al funcției de cost J se poate face utilizând următorul algoritm:

Algorithm 3: Mini-batch Gradient Descent

$w \leftarrow$ *valori random din intervalul* $[-0.1, 0.1]$;
for $epoch = 1 : epochs$ **do**
  $X_{batch} \leftarrow batch\_size$ *linii random din* $\widetilde{X}$;
  $y_{batch} \leftarrow$ *etichetele corespunzatoare liniilor din* X_{batch};
  $w_i \leftarrow w_i - lr \cdot \frac{1}{N}\sum_{j=1}^{batch\_size}(X_{batch}(j,:)\cdot w - y_{batch}(j)) \cdot X_{batch}(j,i)$;
end

Înainte de a aplica algoritmul de mai sus fiecare coloană din $\widetilde{X}$ în afară de ultima (cea de 1-uri) trebuie scalată atât pentru învățare, cât și pentru evaluare, folosind următoarea formulă:

$$col = \frac{col - mean(col)}{std(col)}$$

.

4.3 Cerința

Studentul trebuie să implementeze următoarele funcții:

- `function [w] = learn(X, y, lr, epochs)`
  - Această funcție primește ca parametri o matrice de caracteristici X, un vector coloană de etichete y, un parametru numit rata de învățare lr și un număr de epoci $epochs$ și returnează vectorul de parametri ai modelului w aflat cu ajutorul algoritmului Mini-batch Gradient Descent.
- `function [percentage] = evaluate(path_to_testset, w, histogram, count_bins)`
  - Această funcție primește ca parametri calea către un set de imagini de testare *path_to_testset*, vectorul de parametri ai modelului w, tipul de histograma evaluată ("RGB" sau "HSV") și un număr de valori pentru axa orizontală a histogramei *count_bins* și returnează un procentaj real între 0 și 100 al numărului de imagini clasificate corect *percentage*.

Studentul poate folosi (și i se recomandă) celelalte funcții implementate la partea a treia.

Se dorește o acuratețe de predicție mai mare de **0.55** întrucât setul de date este mult mai mare decât la **part-3**, iar antrenarea se va face *doar* pentru 500 de epoci cu $batch\_size = 64$.

4.4 Punctaj

Toate funcțiile ce trebuie implementate au punctaj egal. Evaluarea se poate face folosind fișierul **checker/checker-part-4/checker_part_4.m** după ce s-au copiat toate sursele necesare în **checker/checker-part-4/**.

5 Precizări

5.1 Trimiterea temei

Studentul va trimite o arhivă pe **vmchecker v2** la **Tema 1** care va conține patru directoare:

- **part-1/** care va conține înăuntrul său sursele implementate pentru partea 1;
- **part-2/** care va conține înăuntrul său sursele implementate pentru partea 2;
- **part-3/** care va conține înăuntrul său sursele implementate pentru partea 3;