1 Introducere

In recunoasterea formelor, selectia si extragerea caracteristicilor reprezinta o alegere decisiva pentru proiectarea oricarui clasificator. Selectia caracteristicilor poate fi vazuta si ca un proces de compresie de date, fiind similara cu o transformare liniara din spatiul initial al observatiilor intr-un spatiu cu mai putine dimensiuni. O astfel de transformare este necesara deoarece poate pastra o mare parte din informatii (prin eliminarea informatiilor redundante sau a celor mai putin semnificative) si permite aplicarea unor algoritmi eficienti intr-un spatiu de dimensiuni reduse.

Cele mai multe transformari utilizate pentru selectia caracteristicilor sunt cele liniare, in timp ce transformarile neliniare au o complexitate mai ridicata, sunt mai dificil de implementat, dar pot avea o eficienta mai mare asupra rezultatelor, exprimand mai bine dependenta dintre formele observate si caracteristicile selectate ale acestor forme.

1.1 Descompunerea valorilor singulare (SVD)

Fiind data o matrice $A \in R^{m*n}$, descompunerea valorilor singulare (in engleza singular value decomposition - SVD) ale matricei A este data de factorizarea $A = USV^T$, unde:

1. $U \in R^{mxm}$ este o matrice ortogonala;
2. $S \in R^{mxn}$ este o matrice diagonala;
3. $V \in R^{nxn}$ este o matrice ortogonala.

Elementele de pe diagonala principala a lui S sunt intotdeauna numere reale non-negative ($s_{ii} \geq 0$ pentru $i = 1 : min(m, n)$) si se numesc *valorile singulare* ale matricei A. Acestea sunt asezate in ordine descrescatoare, astfel incat $s_{11} \geq s_{22} \geq ... \geq s_{rr} > s_{r+1r+1} = ... = s_{pp} = 0$, unde $p = min(m, n)$.

Coloanele $u_j \in R^m, j = 1 : m$ ale lui U se numesc *vectori singulari stanga* ai matricei A. Coloanele $v_j \in R^n, j = 1 : n$ ale lui V se numesc *vectori singulari dreapta* ai matricei A.

De exemplu, pentru matricea:

$$A = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{bmatrix}$$

se obtine urmatoarea descompunere a valorilor singulare:

$$A = USV^T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{bmatrix}$$

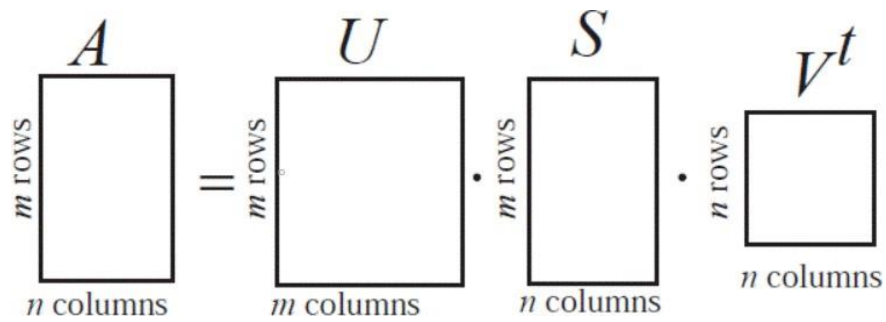

Figura 1: Descompunerea valorilor singulare pentru matricea A de dimensiunem*n, unde $m > n$.

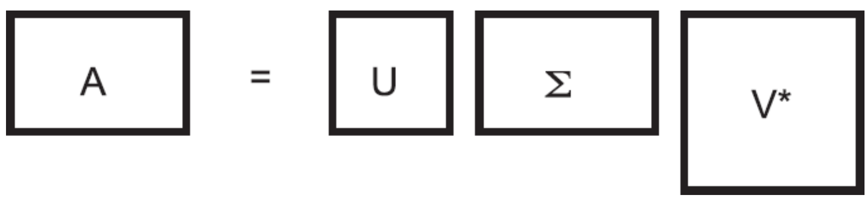

Figura 2: Descompunerea valorilor singulare pentru matricea A de dimensiunem*n, unde $n > m$, $S = \Sigma$, $V^t = V^*$.

2 Compresia imaginilor folosind SVD

Descompunerea redusa a valorilor singulare presupune descompunerea (factorizarea) matricei A astfel: $A \approx A_k = U_k S_k V_k^T$, unde $A_k \in R^{mxn}$, $U_k \in R^{mxk}$, $S_k \in R^{kxk}$, $V_k^T \in R^{kxn}$.

Intuitiv, descompunerea redusa a valorilor singulare semnifica eliminarea valorilor singulare nule sau a valorilor singulare de o valoare foarte mica din matricea S (reprezentand informatia putin semnificativa). Acest lucru presupune si eliminarea coloanelor si a liniilor corespunzatoare acestor valori singulare din matricele U, respectiv din V (vezi Figura 3). Astfel, obtinem o imagine aproximativa care are aproape toata informatia stocata de imaginea originala, doar ca o putem pastra in memorie sub o forma mai restransa.

In cele ce urmeaza, presupunem ca matricea A reprezinta modelarea matematica pentru o imagine alb-negru clara si matricea A_k este modelarea matematica pentru o imagine alb-negru care aproximeaza imaginea clara. Ambele imagini au dimensiune m*n pixeli. Fiecare element (i, j) din matricele A si A_k corespunde intensitatii de gri a pixelului (i, j) din imagine. Prin urmare, elementele matricelor A si A_k au valori cuprinse intre 0 (corespunzatoare culorii negre) si 255 (corespunzatoare culorii albe).

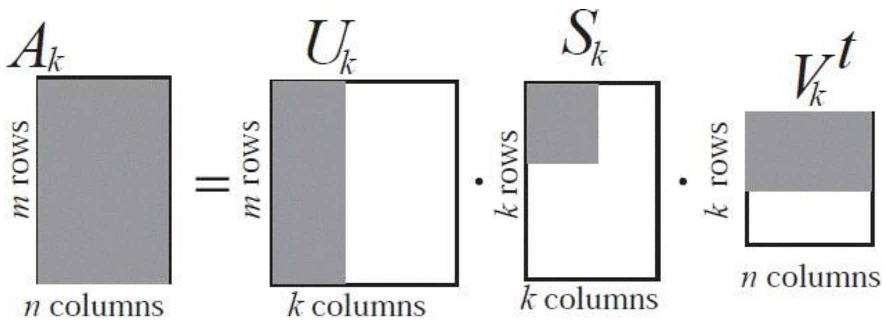

Figura 3: Exemplu de descompunere redusa a valorilor singulare pentru matricea A, m*n dimensionala, m>n. Aceasta descompunere presupune eliminarea portiunilor hasurate in alb din matricele U, S, respectiv V^t. Portiunile hasurate in gri (notate U_k, S_k, respectiv $V_k{}^t$) din matricele U, S, respectiv V se vor pastra. Astfel, matricea A_k va aproxima matricea initiala A.

2.1 Task 1 [20p]

In cadrul acestei cerinte, va trebui sa scrieti o functie Octave pentru compresia unei imagini folosind descompunerea redusa a valorilor singulare. Semnatura functiei este: `function new_X = task1 (photo, k)`, unde *photo* reprezinta imaginea sub forma unei matrici si k numarul de valori singulare. Functia trebuie sa intoarca matricea A_k avand semnificatia de mai sus.

3 Compresia imaginilor folosind analiza componentelor principale

Scopul analizei componentelor principale (in engleza principal component analysis - PCA), este de a transforma date de tipul $A = [a_1, a_2, ..., a_n]$, dintr-un spatiu dimensional R^m intr-un spatiu dimensional R^k, unde $a_i \in R^m$ si k<m. Acest spatiu este dat de cele k componente principale (PC). Componentele principale sunt ortonormate, necorelate si reprezinta directia variatiei maxime. Prima componenta principala reprezinta directia variatiei maxime a datelor, urmand ca urmatoarele componente principale sa aduca variatii din ce in ce mai mici.

O explicatie foarte buna gasiti in urmatorul videoclip pe care vi-l recomand sa il urmariti: https://www.youtube.com/watch?v=TJdH6rPA-TI (poate sa para lung la inceput, dar merita).

3.1 Task 2 [20p]

Urmatorul algoritm calculeaza componentele principale folosind metoda SVD: Fie o matrice $A \in R^{m*n}$. Notam coloanele lui A cu $b_j \in R^{m*1}$ unde $j = 1 : n$ iar liniile lui A cu $a_i \in R^{1*n}$ unde $i = 1 : m$.

1. Se calculeaza media pentru fiecare vector $a_i \in R^{1*n}$, i = 1:m

   $\mu_i = \frac{\sum_{j=1}^{n} a_i(j)}{n}$

   Elementele μ_i formeaza componentele vectorului $\mu \in R^{m*1}$.

2. Se actuailizeaza vectorii $a_i \in R^{1*n}$, i = 1:m astfel: $a_i = a_i - \mu_i$

3. Se construieste matricea $Z \in R^{n*m}$.

   $$Z = \frac{A^T}{\sqrt{n-1}}$$

4. Se calculeaza SVD pentru matricea Z: $Z = USV^T$.

5. Spatiul k-dimensional al componentelor principale (notat cu W) este dat de primele k coloane din matricea $V = [v_1, v_2, ..., v_m]$ astfel:

   $W = [v_1, v_2, ..., v_k]$ (v_1 este prima componenta principala, v_2 este a doua si asa mai departe).

6. Se calculeaza proiectia lui A in spatiul componentelor principale, adica matricea $Y = W^T A$.

7. Se aproximeaza matricea initiala astfel: $A_k = WY + \mu$, unde μ este cel calculat la pasul 1.

Functia Octave care implementeaza aceasta cerinta este: `function new_X = task2 (photo, pcs)`, unde *photo* reprezinta imaginea sub forma unei matrici si *pcs* numarul de componente principale. Functia intoarce matricea new_X care este matricea A_k din algoritmul explicat mai sus.

3.2 Task 3 [20p]

Componentele principale se pot calcula folosind si un algoritm bazat pe matricea de covarianta. Pasii pentru acest algoritm sunt:

Fie o matrice $A \in R^{m*n}$. Notam coloanele lui A cu $b_j \in R^{m*1}$ unde $j = 1 : n$ iar liniile lui A cu $a_i \in R^{1*n}$ unde $i = 1 : m$.

- Pasii 1-2 sunt aceeasi ca la Task-ul 2.
- Se construieste matricea de covarianta $Z \in R^{m*m}$.

  $$Z = \frac{A * A^T}{n-1}$$

- Se calculeaza valorile si vectorii proprii folosind functia *eig* asupra matricei Z: [*V S*] = *eig*(*Z*).

- Spatiul k-dimensional al componentelor principale (notat cu W) este dat de primele k coloane din matricea $V = [v_1, v_2, \ldots, v_m] \in R^{m*m}$:

  $W = [v_1, v_2, \ldots, v_k]$.

- Pasii 6-7 sunt aceeasi ca la cerinta 3.

Functia Octave care implementeaza acesta cerinta este: `function new_X = task3(photo, pcs)`, unde *photo* reprezinta imaginea sub forma unei matrici si *pcs* numarul de componente principale. Functia intoarce matricea new_X care este echivalentul matricei A_k din algoritm.

4 Task 4 - Recunoasterea cifrelor scrise de mana [35p]

Cu ajutorul valorilor si vectorilor proprii se poate rezolva si problema recunoasterii cifrelor. Aceasta problema este un punct de plecare si pentru alti algoritmi asemanatori, cum ar fi recunoasterea scrisului de mana in general. Recunoasterea cifrelor (sau Digit Recognition cum este des intalnita) este folosita de banci in procesarea cecurilor, citirea automata a adreselor persoanelor si citirea automata a formularelor completate. In marea majoritate a cazurilor, problema de fata este una de baza pentru persoanele care lucreaza in domeniul de *machine learning*.

Algoritmul PCA este folosit si el in acest domeniu, de multe ori fiind util pentru a micsora dimensiunea datelor, pastrand ce este mai important din input-ul primit. Astfel, acuratetea nu va avea mult de suferit, dar timpul de procesare al informatiilor va fi semnificativ mai mic, ceea ce este foarte util atunci cand trebuie sa lucrezi cu seturi de date de dimensiuni foarte mari.

Inainte de a prezenta modul in care se va face recunoasterea, mai intai sa vorbim putin de setul de date. „MNIST” este un set de date de 70000 de imagini etichetate, cu cifre scrise de mana colectate de la o multitudine de persoane. Setul are o parte de 60000 de imagini de antrenament si 10000 de imagini care vor fi folosite ca test. Imaginile sunt alb-negru si au o dimensiune de 28x28 pixeli.

Pentru rezolvarea problemei de fata, exista multe variante, cele mai utilizate fiind cele care se folosesc de retelele neuronale. Pentru a va arata cum poate sa arate o astfel de rezolvare, v-am pregatit un program in Python folosind GoogleColab care recunoaste cifrele cu o acuratete de 98.4% ce poate fi gasit la link-ul:
https://colab.research.google.com/drive/1o_O4E-75-G1CFjY599E1Cd5yPcw-XO5s?usp=sharing

Pentru aceasta tema, insa, am propus un algoritm care are la baza PCA-ul si care are o acuratete de aproximativ 93.3%, dar care este mai simplu de inteles si de aplicat decat cele care se bazeaza pe metode de *machine learning*.

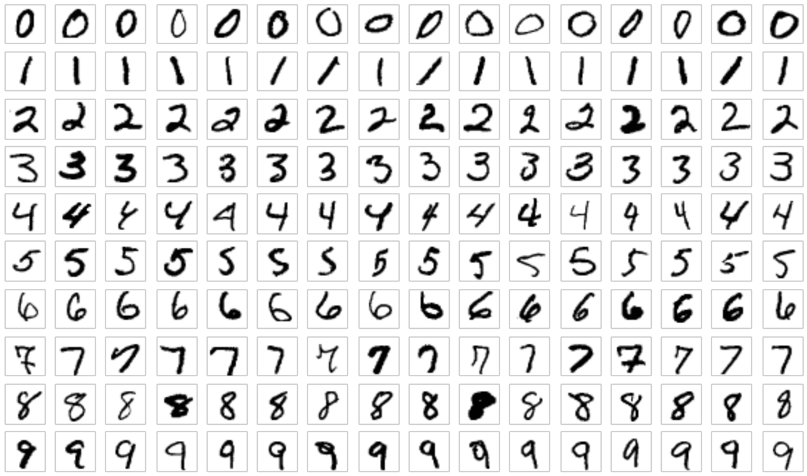

Figura 4 – imagini cu cifre din setul MNIST

Pentru aceasta problema, avand nevoie de multe imagini de antrenament, nu ar fi fost practic sa lucrati cu un folder cu zeci de mii de imagini, asa ca am ales optiunea de a stoca informatiile necesare in fisierul „mnist.mat". Din acest fisier veti avea nevoie doar de imaginile de antrenament. Pentru a lucra cu acest tip de fisier, mai intai trebuie incarcat prin comanda:
d = load ('nume_fisier'). Pentru a obtine datele de antrenament trebuie data comanda X = d.trainX, iar pentru a obtine etichetele care ne indica ce numar este in fiecare imagine folositi comanda y = d.trainY. Astfel, X este o matrice 60000x784, unde pe fiecare linie se afla valorile pixelilor unei imagini, matricea 28x28 fiind transformata intr-un vector de lungime 784.

Dupa ce avem setul de date, se poate incepe propriu-zis lucrul. Pasii algoritmului sunt urmatorii:

1. Pregatirea setului de date prin incarcarea acestora in Octave cu comanda load.
2. Se calculeaza media fiecarei coloane, iar apoi este scazuta din fiecare element al coloanei. Astfel, pixelii fiecarei imagini au fost normalizati.
3. Se calculeaza matricea de covarianta folosind formula cov_matrix = X' * X / (m – 1)
4. Se calculeaza vectorii si valorile proprii ale matricii de covarianta.
5. Se sorteaza valorile proprii in ordine descrescatoare si pastrand aceeasi ordine se sorteaza si vectorii proprii.
6. Vectorii proprii ordonati la pasul 5 sunt folositi pentru crearea unei noi matrici V.
7. Din V sunt selectate primele k coloane. Pentru aceasta versiune a algoritmului k este ales sa fie egal cu 23.
8. Se calculeaza proiectia lui X in spatiul componentelor principale: $Y = X * V_k$
9. Se calculeaza aproximatia lui X notata $\underline{\underline{X}}$ folosind doar 23 de

componente principale: $\underline{X} = Y * V_k'$

10. Se transforma imaginea de test intr-un vector de lungime 784 si se proiecteaza in spatiul componentelor principale prin inmultirea cu V-ul obtinut la pasii anteriori.
11. Se aplica algoritmul k-nearest neighbours pentru care se alege k = 5, iar rezultatul obtinut este predictia cautata.

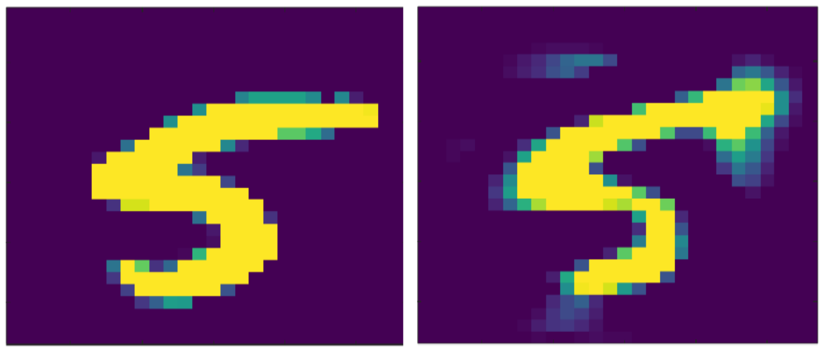

Figura 2 – Cifra initiala si cifra aproximata cu 23 de componente principale

Pentru a calcula predictia finala ne vom folosi de algoritmul k-nearest neighbours care calculeaza cele mai „apropiate" k imagini din setul de antrenament. Alegerea k-ului este o problema in sine deoarece un k prea mic poate duce la gasirea gresita a predictiei, iar un numar prea mare poate duce la suprasaturare de informatii, ceea ce duce, din nou la raspunsuri gresite. Pentru algoritmul de recunoastere a cifrelor prezentat mai sus este suficient un k = 5. Algoritmul k-nearest neighbours are urmatorii pasi:

1. Se foloseste matricea Y care reprezinta proiectia matricii cu datele de intrare in spatiul componentelor principale. Se ia fiecare rand al matricii care reprezinta forma vectorizata a unei imagini si se calculeaza distanta euclidiana dintre acest vector si vectorul de test:
   distance = $\sqrt{(y_1 - x_1)^2 + \cdots + (y_{784} - x_{784})^2}$
2. Se sorteaza crescator distantele obtinute si se pastreaza doar primele 5 si se pastreaza intr-un vector valorile care reprezinta ce numar era in imaginea respectiva (i.e. prima poza din set este un 5, a doua este un 0 etc.).
3. Se calculeaza mediana vectorului obtinut mai sus si se obtine astfel predictia (HINT: functia median din Octave).

Pentru acest Task veti avea de implementat in total 6 functii: *prepare_data, visualise_image, magic_with_pca, prepare_photo, KNN* si *classify_image*.

Functia prepare_data are semnatura: *[train_mat, train_val] = prepare_data (name, no_train_images)* si cu ajutorul ei veti importa datele necesare din fisierul „mnist.mat".

Functia visualise_image are semnatura: *im = visualise_image (train_mat, number)* si cu ajutorul ei puteti vizualiza imaginea cu numarul number din setul de date. Daca decomentati ultima comanda, atunci puteti vedea imaginea cu care lucrati.

Functia magic_with_pca are semnatura: *[train, miu, Y, Vk] = magic_with_pca (train_mat, pcs)* si aplica PCA asupra matricii de antrenament, implementand pasii 2-9 din algoritmul de predictie.

Functia prepare_photo are semnatura: *sir = prepare_photo (im)* si primeste imaginea de test pe care o modifica si o transforma intr-un sir pentru a se putea face mai usor predictia. Imaginile de antrenament au fundalul negru si cifra alba, pe cand cele de test au fundalul alb si cifra neagra, asa ca trebuie sa se inverseze culorile.

Functia KNN are semnatura: *prediction = KNN (labels, Y, test, k)* si pune aplica algoritmul de k-nearest neighbours.

Functia classify_image are semnatura: *prediction = classifyImage (im, train_mat, train_val, pcs)* si pune cap la cap functiile anterioare pentru a returna predictia pe care o face.

Testele sunt impartite pe 4 categorii: „data image", „data processing image", „invert image" si „prediction".

- Pentru a trece oricare dintre teste, mai intai trebuie sa implementati functia „visualise_image".
- Pentru a trece testele „data image" trebuie implementata si functia *„prepare_data"*
- Pentru a trece testele „data processing image" mai trebuie implementata si functia *„magic_with_pca"*
- Pentru a trerce testele „invert image" trebuie implementata functia *„prepare_photo"*
- Pentru a trece testele „prediction" trebuie implementate toate functiile

Figura 3 – Modul in care sunt pastrate in memorie datele de antrenament (stanga) si modul in care sunt pastrate imaginile de test (dreapta)