Lycée Lakanal

2025- 2026

PC

Devoir d'informatique

Les quatre parties de ce problème sont très indépendantes :

La partie **II** dépend très peu de la partie **I** ;

Les parties **III** et **IV** sont complètement indépendantes du reste.

Reconnaissance de panneaux routiers (puis on part au ski)

I Algorithme KNN

Dans cette partie, nous allons développer un algorithme simple capable de reconnaitre une image automatiquement à partir de l'apprentissage d'une base d'images sources. Pour cela, nous allons utiliser l'algorithme KNN, dit de recherche des « k plus proches voisins » en utilisant la norme euclidienne.

Nos images recherchées seront supposées prétraitées afin qu'elles ressemblent à cela :

Le prétraitement numérique aura donc réalisé les étapes suivantes :

- Transformation projective en couleur afin d'avoir une image « vue de face » et cadrée.
- Redimensionnement des images avec 100 lignes et 100 colonnes, soit 10 000 pixels RGB.
- Enregistrement du résultat au format BMP.

Les images sources permettant l'apprentissage auront subi un traitement supplémentaire de mise en blanc du fond en dehors du panneau.

On supposera que les panneaux étudiés pourront être de p types différents (stop, priorité à droite, sens interdit…). Ces p types seront numérotés de 0 à $p-1$.

On supposera les entiers p et k entrés.

Mise en forme préliminaire

Chacune des images considérées est représentée par un tableau numpy (**numpy.ndarray**) ayant 100 lignes et 100 colonnes, soit 100×100 pixels. Chaque pixel d'une image est représenté par un triplet (R, G, B) de trois flottants (représentant respectivement les niveaux de rouge, de vert et de bleu du pixel).

Pour chaque image, on désire créer une liste **L_RGB** des valeurs de R, G et B de chacun de ses pixels (exemple sur un parcours ligne par ligne : L_RGB $= \left[R_{0,0}, G_{0,0}, B_{0,0}, R_{0,1}, G_{0,1}, B_{0,1}, ... \right]$. Ainsi, avec des images ayant toujours la même dimension de 100×100 pixels, on obtient une liste de 30 000 valeurs pour chacune.

Q1. Créer une fonction **Analyse(Image)** qui, à partir d'une image donnée sous forme d'array, renvoie la liste L_RGB associée.

On dispose d'un jeu d'entraînement, composé des deux données suivantes :
- Une liste **L_entr** de N photos (tableaux numpy de taille 100×100 **)** représentant des panneaux « modèles ».
- Une liste **Num_entr** de N entiers, indiquant à quel type de panneau chaque photo de la liste **L_entr** correspond. Par exemple, si Num_entr$[8]$ est égal à 13, cela signifie que la neuvième photo du jeu d'entraînement, c'est-à-dire L_entr$[8]$, représente un panneau de type 13.

On dispose également d'un jeu test, construit sur le même modèle, et constitué des deux données suivantes :
- Une liste **L_test** de M photos représentant des panneaux.
- Une liste **Num_test** de M entiers, indiquant à quel type de panneau chaque photo de la liste **L_entr** correspond.

Q2. Ecrire un code Python qui crée deux listes **RGB_entr** et **RGB_test**, où :
- **RGB_entr** est la liste des listes **L_RGB** associées à chacun des éléments de **L_entr** ;
- **RGB_test** est la liste des listes **L_RGB** associées à chacun des éléments de **L_test** .

Quelques fonctions annexes utiles

Q3. Ecrire une fonction **maxi** ayant pour paramètre d'entrée une liste de flottants, renvoyant le maximum de cette liste, ainsi que l'indice où il est atteint. En cas d'égalité, on renverra le premier indice où ce maximum est atteint : par exemple, maxi $\left(\left[1, 3, 7, 5, 6, 7, 3, 1 \right] \right)$ devra renvoyer $\left(7, 2 \right)$.

Q4. Ecrire une fonction **majoritaire** ayant pour paramètre d'entrée une liste de flottants L, et renvoyant le réel qui est présent le plus souvent dans L. Là en encore, en cas d'égalité, on renverra le premier des réels les plus fréquents. Ainsi, majoritaire $\left(\left[1, 3, 2, 5, 3, 7, 3, 1 \right] \right)$ renvoie 3 et majoritaire $\left(\left[1, 3, 2, 5, 3, 1, 3, 1 \right] \right)$ renvoie 1.

Remarque : *on pourra éventuellement utiliser la fonction précédente*.

Mise en place de l'algorithme KNN

Chaque image est désormais représentée par un n – uplet (modélisé par une liste de n termes). On suppose que le nombre n de termes de chaque n – uplet est identique dans tout ce sujet.

On rappelle que la distance euclidienne $d\left(u, v \right)$ entre deux n-uplets $u = \left(u_0, u_1, ..., u_{n-1} \right)$ et $v = \left(v_0, v_1, ..., v_{n-1} \right)$ est donnée par : $d\left(u, v \right) = \sqrt{\sum_{i=0}^{n-1} \left(u_i - v_i \right)^2}$.

Q5. Créer une fonction **Distance_uv(u,v)** calculant la distance entre deux n – uplets.

Q6. Créer une fonction **Distance_totale(u, Lv)** , ayant pour paramètres d'entrée un n – uplet u, et une liste L_v de n – uplet, renvoyant une liste des distances euclidiennes entre u et chacun des n – uplets de la liste L_v, sous la forme :

$\left[\left[\text{ distance entre } u \text{ et } v, \text{ indice de } v \text{ dans } L_v \right], \text{ pour tout } v \in L_v \right]$.

Q7. Ecrire une fonction **Tri(L)** ayant pour paramètre d'entrée une liste $L = \left[\left[a_0, b_0 \right], \left[a_1, b_1 \right], ..., \left[a_{p-1}, b_{p-1} \right] \right]$ de p sous-listes de deux flottants, et renvoyant la liste des mêmes sous-listes triée dans l'ordre des premières composantes croissantes.

La fonction devra donc renvoyer une liste de la forme $L\text{triée} = \left[\left[a_{i_0}, b_{i_0} \right], \left[a_{i_1}, b_{i_1} \right], ..., \left[a_{i_{p-1}}, b_{i_{p-1}} \right] \right]$, avec $a_{i_0} \leq a_{i_1} \leq ... \leq a_{i_{p-1}}$.

On pourra adapter l'algorithme de tri que l'on souhaite. On en donnera le nom, ainsi que l'ordre de grandeur de sa complexité.

Q8. Créer une fonction **Proches(u, Lv, k)** qui renvoie une liste des k plus proches voisins de u dans la liste L_v au sens de la norme euclidienne. Pour préciser, cette fonction devra renvoyer une liste de k listes de la forme $\left[\text{distance}, \text{indice} \right]$.

Remarque : On supposera que k est plus petit que le nombre de n - uplets de L_v.

Q9. A l'aide des questions **Q6.**, **Q4.**, des listes **Num_entr** et **RGB_entr**, écrire une fonction **decision(u, k)** ayant pour paramètre d'entrée un n – uplet u (représentant un panneau), et renvoyant le numéro du type de panneau modèle auquel l'algorithme des k plus proches voisins lui fait correspondre.

Q10. En déduire un code créant une liste **Num_test_KNN** de même longueur M que **L_test**, et telle que, pour tout entier $0 \leq i \leq M - 1$, $\text{Num\_test\_KNN}\left[i \right]$ est le numéro du type de panneau modèle que l'algorithme des k plus proches voisins fait correspondre à l'image numéro i du jeu de test.

Pour évaluer l'efficacité de l'algorithme **KNN**, on utilise une matrice de confusion : On note $M = \left(m_{i,j} \right)_{0 \leq i,j \leq p-1}$ la matrice définie par : pour tout $\left(i, j \right) \in \left\{ 0, ..., p - 1 \right\}^2$, $m_{i,j}$ est le nombre de fois où, lors du test effectué en **Q10.**, une image, représentant réellement un panneau de type j, a été diagnostiqué par l'algorithme comme représentant un panneau de type i. Ainsi la matrice M est diagonale si et seulement si l'algorithme a été entièrement fiable (un panneau de type j a toujours été diagnostiqué comme étant un panneau de type j).

Q11. On suppose que $p = 5$, que **Num_test_KNN** $= \left[1, 0, 2, 3, 0, 1, 4, 2, 1, 4 \right]$ et que **Num_test** $= \left[0, 1, 2, 3, 0, 2, 4, 1, 2, 4 \right]$. Donner la matrice M correspondante.

Q12. Ecrire une fonction **Confusion(p, L1, L2)**, ayant pour paramètres d'entrées l'entier p ainsi que deux listes **L1** et **L2** correspondant respectivement aux listes **Num_test** et **Num_test_KNN**, et qui renvoie la matrice M.

Q13. Ecrire une fonction **Taux** ayant pour paramètre d'entrée la matrice M, et renvoyant en retour une liste de couples $L = \left[\left(pos_0, neg_0 \right), ..., \left(pos_{p-1}, neg_{p-1} \right) \right]$, où pour tout $i \in \left\{ 0, ..., p - 1 \right\}$:

pos_i est le taux de vrais positifs pour le type de panneau i : pourcentage du nombre panneaux de type i ayant été repérés comme étant de type i, les panneaux étant (réellement) de type i ;

neg_i est le taux de vrais négatifs pour le panneau i : pourcentage du nombre de panneaux n'ayant pas été repérés comme étant de type i, pami les panneaux n'étant (réellement) pas de type i.

II Algorithme des p moyennes (bon normalement c'est k moyennes, mais là c'est p).

On rappelle le principe de l'algorithme des p moyennes :

A partir d'un ensemble de N vecteurs $U = [u[0], ... u[N-1]]$, l'algorithme vise à les partitionner en p clusters $[S[0], ..., S[p-1]]$ de centres respectifs $[c[0], ... c[p-1]]$ avec $p \leq N$ de la manière suivante :

Choix initial de p centres $c[i]$ (chaque $c[i]$ étant un vecteur).

- Etape 1 : En notant x_i^j les n_j points du cluster S_j, cette étape vise à affecter les N points x_i dans p clusters S_j tels que $\text{distance}\left(x_i^j, c_j\right) = \min_k \left(\text{distance}\left(x_i^j, c_k\right)\right)$.
- Etape 2 : Calcul des nouveaux centres c_j des p clusters S_j : $c_j = \frac{1}{n_j}\sum_{i=1}^{n_j} x_i^j$

- Itérations : Répétition des étapes 1 et 2 jusqu'à ce que les $[c[0], ... c[p-1]]$ n'évoluent plus.

Q.14. Ecrire une fonction **barycentre(L)** ayant pour paramètre d'entrée une liste $L = [L[0], ..., L[N-1]]$ de N vecteurs $L[0], \ldots, L[N-1]$ (représentés par des array numpy) tous de même taille n, et renvoyant le barycentre de ces vecteurs.

On rappelle également que l'on dispose d'une fonction **Distance_uv(u,v)** calculant la distance entre deux vecteurs de même longueur.

Q15. Ecrire une fonction **PlusProches(u , S)** , ayant pour paramètres d'entrée un vecteur u de longueur n, une liste S de p vecteurs tous de longueur n, et renvoyant en retour l'indice $0 \leq i \leq p-1$ du vecteur $S[i]$ de la liste S qui est le plus proche de u.

Q16. Ecrire une fonction **Repartition(U , S)** , ayant pour paramètres d'entrées une liste U de N vecteurs tous de même longueur n, une liste S de p vecteurs tous de même taille également n, et qui renvoie une liste P de longueur N, où, pour tout i tel que $0 \leq i \leq N-1$, $P[i]$ est l'indice du vecteur de S qui est le plus proche du vecteur $U[i]$ de U.

Q17. Ecrire une fonction **nouveaux_barycentres(p, U , P)** . Cette fonction a pour paramètres d'entrées :

- Un entier p ;
- U, liste de N vecteurs tous de même longueur ;
- P, liste de N entiers tous compris entre 0 et $p-1$.

Cette fonction doit renvoyer une liste $S = [S[0], ..., S[p-1]]$ de p vecteurs, telle que, pour tout $0 \leq i \leq p-1$, $S[i]$ est le vecteur barycentre de ceux des vecteurs $U[j]$ de U qui sont tels que $P[j] = i$.

Q18. Déduire de ce qui précède une fonction **pMoyennes(U , C , p)** , appliquant l'algorithme des p moyennes aux vecteurs de U (liste de N vecteurs), pour un choix de centres initiaux $C = [c[0], ... c[p-1]]$.

Cette fonction doit renvoyer la liste P des indices des classes auxquelles sont affectées les éléments de U après exécution de l'algorithme.

On rappelle que l'on dispose de la liste **RGB_entr** : liste correspondant à photos de panneaux, chacune de ces photos étant représentée dans la liste **RGB_entr** par la sous-liste RGB_entr $[k]$, cette sous-liste étant de longueur 30000 .

On dispose également de la liste d'entiers **Num_entr** . Elle est de même longueur N que **RGB_entr** , et, pour tout $k < N$, Num_entr $[k]$ est le numéro du type de panneau que représente la photo représentée par la liste RGB_entr $[k]$.

Q19. Ecrire une fonction **Initialisation_barycentres**, ayant pour paramètres d'entrée les listes **Num_entr** et **RGB_entr** , ainsi que

l'entier **p**. Cette fonction renverra une liste C de longueur p telle que, pour tout i entier tel que $0 \leq i \leq p-1$, $C[i]$ est le barycentre des éléments de la liste **RGB_entr** qui représentent le panneau de type i.

Q20. Quel but pourrait avoir la suite d'instructions :

```
C = Initialisation_barycentres(Num_entr , RGB_entr,p)
P = pMoyennes(RGB_test , C, p)
```

Commentez (et critiquez amplement !).

III Exploitation d'une base de données

On dispose d'une base de données comportant deux tables dont voici des extraits :

panneaux

num	type	num_entreprise	efficacité	date
1	*A*	119	0,95	1988
2	*AB*	27	0,76	2019
3	*A*	27	0,84	2013
4	*D*	34	0,97	2005
5	*AB*	22	0,85	2021

entreprises

num	nom
1	Abeon
2	ARP signal
3	Irosign

Q21. num entreprise est une clé étrangère de la table panneaux. Expliquer.

On demande de répondre aux questions Q22 à Q27 suivantes à l'aide d'une et d'une seule requête SQL par question.

Il n'est pas interdit d'expliquer son raisonnement.

Q22. Ecrire une requête permettant de déterminer combien de panneaux de type "AB " sont présents dans la table panneaux.

Q23. Ecrire une requête permettant de déterminer quelle est l'efficacité moyenne d'un panneau de type "AB ".

Q24. Quel est le type de panneaux le plus représenté dans la table panneaux ?

Q25. Quel est le type de panneaux ayant la plus mauvaise efficacité moyenne ?

Q26. Ecrire une requête donnant la liste des noms des entreprises fournissant des panneaux de type "AB ".

Q27. Ecrire une requête donnant le nom de l'entreprise ayant la plus mauvaise efficacité moyenne, parmi celles qui fournissent des panneaux de type "AB ".

IV Plus de panneaux : du ski et un peu de programmation dynamique pour finir

N skieurs rentrent dans un magasin pour louer N paires de skis (parmi $M > N$ paires disponibles. On souhaite leur donner à tous une paire qui leur convient au mieux (*On suppose qu'un skieur sans paire de ski correspond au cas où la paire de skis est de longueur nulle.).*

On suppose le meilleur choix consiste dans le fait que la longueur de la paire de skis doit être au plus proche de la taille du skieur.

On cherche donc à minimiser la quantité : $\rho(\sigma) = \sum_{i=0}^{N-1} \left| t_i - s_{\sigma(i)} \right|$, où :

- $t_0, t_1, ..., t_{N-1}$ sont les tailles des N skieurs, représentées par une liste $T = [T[0], ..., T[N-1]]$ de taille N.
- $s_0, s_1, ..., s_{M-1}$ sont les longueurs des paires de skis, représentées par une liste $S = [S[0], ..., S[N-1]]$ de taille M.
- σ est une injection de $\{0, ..., N-1\}$ dans $\{0, ..., M-1\}$, représentée par une liste $[\sigma[0], ..., \sigma[N-1]]$,

de taille N, avec $0 \leq \sigma[0], ..., \sigma[N-1] \leq M-1$. Dans cette représentation, $\sigma[i] = j$ correspond au fait d'attribuer au skieur i la paire de ski j.

Q28. On suppose que $T = [1, 1.5, 2]$, que $S = [0.75, 0.83, 1.25, 1.75, 1.9, 2.05, 2.2]$ et que $\sigma = [1, 3, 5]$.

Calculer $\rho(\sigma)$.

Q29. Ecrire une fonction **rho(T,S, Sigma)** , ayant pour paramètres d'entrées trois listes **T**, **S**, **Sigma** correspondant aux descriptions de T, S, σ faites ci-dessus, et renvoyant la quantité $\rho(\sigma)$.

On suppose désormais ordonnés les paires et les skieurs par tailles croissantes : $t_0 < t_1 < ... < t_{N-1}$ (tailles des skieurs), et $s_0 < s_1 < ... < s_{M-1}$ (longueur des skis). Résoudre le problème revient à prendre les skieurs dans l'ordre croissant et à les placer en face d'une paire de skis dans l'ordre où elles viennent. C'est comme si on insérait des espaces dans la séquence des skieurs sans en changer l'ordre :

t_1		t_2	t_3			t_4	...	t_{N-1}		t_N	
s_1	s_2	s_3	s_4	s_5	s_6	s_7	...	s_{M-3}	s_{M-2}	s_{M-1}	s_M

.

Q30. Expliquer pourquoi l'algorithme suggéré ci-dessus permet d'obtenir la solution optimale.

On définit : $p(n, m) = \sum_{i=0}^{n-1} \left| t_i - s_{\sigma_{n,m}(i)} \right|$, où $\sigma_{n,m}$ désigne le meilleur choix possible, pour les n premiers skieurs, de n paires de skis parmi les m premières (on ne considère que les n premiers skieurs et les m premières paires de skis, et l'on fait le meilleur choix pour ces données).

Q31. Que vaut $p(0, m)$ pour $0 \leq m \leq M$? Que vaut $p(n, 0)$ pour $0 \leq n \leq N$?

Q32. Pour $1 \leq n \leq N$ et $1 \leq m \leq M$, exprimer $p(n, m)$ en fonction de $p(n, m-1)$, de $p(n-1, m-1)$ et de t_n et s_m .

Q33. Ecrire une fonction **choix_skis** $(\mathbf{T}, \mathbf{S})$, ayant pour paramètres d'entrées deux listes T et S triées, correspondant aux tailles des skieurs et des skis, et (avec les notations précédentes) renvoyant $\rho(\sigma) = \sum_{i=0}^{N-1} \left| t_i - s_{\sigma(i)} \right|$ pour le meilleur choix σ possible.

Q34. Comment modifier la fonction précédente de manière à ce qu'elle indique en plus quel choix σ faire ?

(*On pourra réécrire la fonction, ou décrire des pistes de ce qu'il faudrait faire…).*

FIN