

TP APPRENTISSAGE

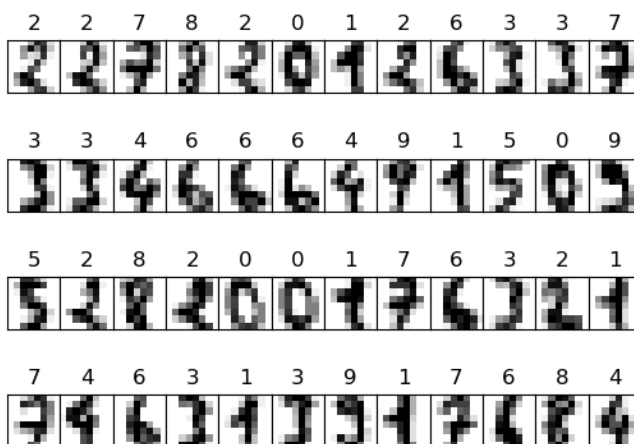
Commencer par télécharger le fichier `TP_apprentissage.zip` sur le site du cours. Dézipper le fichier (clic droit puis extraire ici) et ouvrir le fichier `TP_apprentissage.py`.

Ensuite, modifier l'adresse du dossier `TP_apprentissage/` dans la commande `os.chdir`. Si votre IDE exécute le script Python dans le dossier dans lequel il se trouve, vous pouvez simplement commenter cette ligne. Exécuter le fichier.

- Si le message vous dit qu'il n'y a pas de problème, commencer le TP ;
- sinon suivre la consigne du message d'erreur.

Première partie

L'algorithme des k plus proches voisins pour la reconnaissance de chiffres manuscrits



Dans le dossier `TP_apprentissage/` se trouvent 200 images de chiffres sous la forme `Image_xxx_y.png`, où `xxx` est un nombre compris entre 000 et 199, et `y` est un chiffre entre 0 et 9 (le chiffre représenté par l'image).

Le fichier `TP_apprentissage.py` fournit les fonctions suivantes :

- `image2liste(nom_image)` renvoyant une liste de liste d'entiers correspondant aux pixels l'image `nom_image` codés sous formes de simples entiers car l'image est en niveau de gris (ici entre 0 et 255). L'image a été aplatie : elle est représentée par une simple liste, plutôt que par une matrice, obtenue en concaténant les lignes de la matrice.
- `base_images_supervise()` renvoyant une liste de 200 couples (`image`, `chiffre`) dont le premier élément `image` est la liste correspondant à l'image aplatie de la fonction précédente et le deuxième élément `chiffre` est le chiffre représenté sur l'image.
- `affiche(image, titre="")` affichant une image donnée sous forme de liste avec un titre éventuel.

Le but de cette partie est d'utiliser l'algorithme des k plus proches voisins (k -NN) pour identifier le chiffre représenté par une image.

Dans cette partie, on utilisera les listes suivantes :

Python

```
1 Z = base_images_supervisee() # base de donnée complète
2 X = Z[:150] # base d'apprentissage
3 T = Z[150:] # base de test
```

Distance entre deux images

On utilise la distance euclidienne entre deux images « aplaties » de taille dim :

$$d(I, I') = \sqrt{\sum_{k=0}^{dim-1} (I_k - I'_k)^2}.$$

1. Écrire une fonction `distance2(image1, image2)` renvoyant le carré de la distance entre deux images.

Python

```
1 >>> distance2(base[0][0], base[1][0])
2 5698302
```

Algorithme des k plus proches voisins

2. Écrire une fonction `voisins_tries(X, image)` prenant en entrée
 - une liste X constituée de couples $(image_x, chiffre)$ où $chiffre$ est le chiffre représenté sur l'image $image_x$;
 - une image $image$

et renvoyant une liste de couples $(dist, chiffre)$ où, pour chaque image $image_x$ contenue dans la liste X , $dist$ est la distance (au carré) de l'image $image$ à l'image $image_x$ et $chiffre$ est le chiffre représenté sur l'image $image_x$.

On demande de plus que cette liste soit triée par distances croissantes. On pourra pour cela appliquer la fonction `sorted` à la liste des couples (ce qui correspond, sans précision supplémentaire, à l'ordre lexicographique sur les couples : il nous convient ici.)

3. Écrire une fonction `k_voisins(X, k, image)` prenant les mêmes entrées que la fonction précédente ainsi qu'un entier un entier $k < \text{len}(X)$, comptant le nombre de chaque chiffre apparaissant parmi les k plus proches voisins (on pourra utiliser une liste `compteur` telle que `compteur[i]` compte le nombre de fois où le chiffre i apparaît) puis renvoyant le chiffre le plus présent.

Tester avec

Python

```
1 for i in range(10): # Des réussites et des échecs...
2     titre = f"Chiffre deviné : {k_voisins(X, 10, T[i][0])} - chiffre réel : {T[i][1]}"
3     affiche_image(T[i][0], titre)
```

Matrice de confusion

Pour mesurer les erreurs commises par notre algorithme de classification supervisée, et choisir la valeur de k de manière optimale, on utilise la base de test T et on compte le nombre d'images pour lesquelles la prédiction est erronée.

Pour cela on construit la **matrice de confusion** C de taille 10×10 dont les coefficients $c_{i,j}$ correspondent aux nombres d'images de T représentant le chiffre i et pour lesquelles l'algorithme a trouvé le chiffre j .

Plus la matrice est proche d'une matrice diagonale, plus le choix de k utilisé est pertinent.

4. Écrire une fonction `matrice_confusion(X, T, k)` renvoyant la matrice de confusion (une liste de listes d'entiers) associée à ses arguments.

Python

```
1 >>> np.array(matrice_confusion(X, T, 12)) # np.array juste pour l'affichage !
2 [[3 0 0 0 0 0 0 0 0 0]
3  [0 4 0 0 0 0 0 0 0 0]
4  [0 2 4 0 1 0 0 0 1 1]
5  [0 0 0 4 0 0 0 0 0 0]
6  [0 1 0 0 3 0 0 0 0 1]
7  [0 2 0 1 0 1 0 0 0 0]
8  [0 1 0 0 0 0 3 0 0 0]
9  [0 2 0 0 0 0 0 3 0 0]
10 [0 1 0 0 1 0 0 0 1 0]
11 [0 0 0 0 0 0 0 0 1 8]]
```

5. Écrire une fonction `somme_non_diagonal(C)` renvoyant la somme des valeurs non diagonales C (donc des erreurs commises par notre algorithme).

Python

```
1 >>> somme_non_diagonal(matrice_confusion(X, T, 5))
2 11
```

6. Compléter la fonction `meilleur_k(X, T, k_max)` calculant les matrices de confusion pour $k \in \llbracket 1, k_{\max} \rrbracket$, affichant la courbe du pourcentage d'erreurs en fonction de k et renvoyant la valeur de k qui le minimise.
Tester avec `meilleur_k(X, T, 15)`.

Remarque importante

Le fait que les pourcentages d'erreur varient selon k ne dit **rien** d'autre que ceci : modifier k a une influence sur la qualité du modèle (ce serait aussi le cas en changeant de distance.)

On pourrait être tenté de choisir k minimisant l'erreur sur la base de test, mais cela ne signifie pas que l'on minimise alors l'erreur du modèle, globalement.

Choisir le meilleur k à partir de la base de test revient en quelque sorte à utiliser la base de test comme une deuxième base d'entraînement.

Pour estimer la qualité du modèle, il faut utiliser des données n'ayant pas servi à construire ce modèle.

Comment trouver une bonne valeur de l'**hyperparamètre**^a k ?

Il faudrait trois jeux de données :

- une **base d'apprentissage** servant à construire des modèles pour différentes valeurs de l'hyperparamètre k ;
- une **base de validation** pour laquelle on calcule le taux d'erreurs de chacun de ces modèles ;
- une **base de test** pour tester le modèle avec le taux d'erreurs le plus faible.

a. Nom d'un paramètre n'étant pas déterminé par la base d'apprentissage.