2MA100 - Devoir Maison 2

Les exercices de ce devoir maison noté sur 20 points doivent être rendus au choix au format **Notebook** (`.ipynb`) ou **Script** (`.py`) en un ou plusieurs fichiers au plus tard le **29 avril 2022** à 20h00 sur Moodle.

Attention Plagiat.

Les devoirs maisons doivent être codés et rendus de manière individuelle. A titre d'information, voici une définition du plagiat adaptée du Memento de l'Université de Genève :

Le plagiat consiste à insérer, dans un travail académique, des formulations, des phrases, des passages, des morceaux de code, des images, de même que des idées ou analyses repris de travaux d'autres auteurs, en les faisant passer pour siens.

En particulier, le copier-coller à partir de sources trouvées sur Internet ou sur des travaux d'autres étudiant·es sans citer les sources est considéré comme du plagiat et implique une note zéro. Le plagiat constitue également une tentative de tricherie sanctionnée par le règlement de l'université. La solution est d'indiquer dans vos devoirs tout de ce qui ne vient pas de vous en mentionnant les sources (page Internet, livres, autre étudiant·e,...). Tous les fichiers rendus seront analysés automatiquement avec un logiciel de détection des similarités (entre étudiant·es et depuis Internet).

Les modules Numpy et Matplotlib peuvent être utiles dans certains exercices :

```
import numpy as np
import matplotlib.pyplot as plt
```

Exercice 1 : Reconnaissance de chiffres manuscrits

Le but de cet exercice est d'écrire un programme de classification d'image de chiffres manuscrits. Ceci fut l'une des premières applications industrielles du machine learning à la lecture automatique des chèques ou des codes postaux.

Les instructions suivantes permettent de charger un jeu de données de chiffres manuscrits numérisés disponible dans le package `scikit-learn` (nom d'import `sklearn`) :

```
from sklearn.datasets import load_digits
digits = load_digits()
X, y = digits.data, digits.target
```

Dans le cas où le module `sklearn` est indisponible, il est possible utiliser les commandes suivantes à la place en ayant pris soin de placer le fichier `digits.csv` dans le répertoire courant :

```
digits = np.loadtxt("digits.csv", delimiter =',')
X, y = digits[:,:-1], digits[:,-1]
```

Ainsi `X` est un tableau Numpy qui contient de nombreux exemples de chiffres manuscrits numérisées en image de 8x8 pixels stockés sous la forme de tableau de 64 nombres entiers stockés en flottants. La variable `y` contient l'entier entre 0 et 9 correspondant au chiffre numérisé. On parle de *label*.

a) Quelle commande Python permet de connaître les dimensions de `X` et `y` et ainsi de connaître le nombre d'exemples contenus dans la base de données ?

b) Afficher à l'aide de la commande `print` les données contenus dans `X` associées à l'indice `idx=12` ? Il s'agit donc de la douzième ligne du tableau `X`.

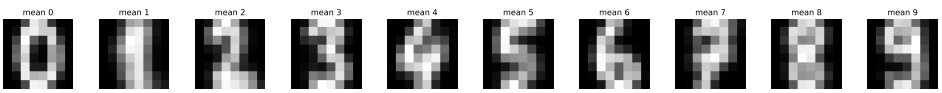
c) À l'aide des fonctions `reshape` de Numpy et `imshow` de Matplotlib afficher l'image d'indice `idx=12`. Il est possible d'utiliser l'argument `cmap='gray'` dans l'appel de `imshow` pour afficher le résultat en niveau de gris. Quel chiffre est ainsi codé ?

Pour chacune des classes de chiffre (de 0 à 9), on souhaite calculer son centroïde, *i.e.* la représentation "moyenne" d'une classe.

d) Comment définir les sous-tableaux de `X` et de `y` correspondant à tous les chiffres 0 numérisés ?

e) Pour l'ensemble des 0 de la question précédente, calculer pour chaque pixel la valeur moyenne, afin de définir le "zéro moyen".

f) Pour l'ensemble des chiffres de 0 à 9 tracer sur une même ligne (à l'aide de la fonction `subplot` de Matplotlib et en initialisant la figure avec `plt.figure(figsize=(20,2))`) l'image moyenne associée :



Finalement, nous allons implémenter notre propre classifieur : pour une nouvelle image de chiffre numérisé, on prédit la classe dont le chiffre moyen est le plus proche. Pour cela on partage notre jeu de données en deux parties de tailles semblables : la première partie servira de données d'entraînement (`X_train` et `y_train`) ; la seconde partie servira de données de tests (`X_test` et `y_test`).

g) Définir les variables : `X_train`, `y_train`, `X_test` et `y_test`.

h) Pour chaque chiffre de l'ensemble d'entraînement, calculer les centroïdes (*i.e.* les chiffres moyens) des classes de 0 à 9. On notera la variable contenant l'ensemble des moyennes `centroids_train`.

i) Pour chaque chiffre de l'ensemble de test (`X_test`), calculer le centroïde appartenant à `centroids_train` le plus proche (dans la norme euclidienne).

j) Finalement, évaluer si le chiffre ainsi obtenu correspond au vrai chiffre en utilisant `y_test` et en déduire une estimation du pourcentage de bonnes prédictions sur l'ensemble de test.

Exercice 2 : Algorithme PageRank

Le but de cet exercice est d'implémenter l'algorithme PageRank. Il est interdit d'utiliser le module NetworkX (sauf pour vérifier vos implémentations). On considère un ensemble de pages web et on veut les classer par ordre d'importance. Pour cela on considère un graphe orienté $G = (X, E)$ dont les noeuds représentent les pages web. Une arête va d'un noeud a vers un noeud b s'il existe un lien sur la page a qui mène vers b. Les liens d'une page vers elle même sont pas considérés. Un tel graphe orienté peut être représenté sous forme de dictionnaire, dont les clefs sont un ensemble de pages internet ; la valeur associée à chaque clef est l'ensemble des liens présents sur cette page.

Les questions **a)**, **e)** et **f)** ne nécessitent pas d'implémentation Python, elle peuvent donc être résolues au choix :

- sur papier ; dans ce cas scanner ou prendre en photo vos réponses et mettre les fichiers correspondant sur Moodle
- dans le notebook contenant vos implémentations en utilisant LaTeX
- avec le logiciel de votre choix ; dans ce cas déposer sur Moodle un fichier PDF

a) Dessiner à la main le graphe associé au dictionnaire suivant :

```
dic = {'mathématiques' : {'algèbre', 'analyse', 'arithmétique'}, \
'arithmétique' : {'algèbre', 'mathématiques', 'cryptographie'}, \
'algèbre' : {'analyse', 'mathématiques'}, \
'analyse' : {'algèbre', 'mathématiques'},
'cryptographie': set()}
```

b) Dans la suite de cet exercice les graphes seront toujours orientés et sans boucles et représentés sous forme de dictionnaires avec des noeuds numérotés de 0 à $n-1$ où n désigne le nombre de noeuds. Définir le graphe précédent sous forme d'un dictionnaire avec les noeuds numérotées de 0 à 4.

c) La matrice d'adjacence d'un graphe est une matrice carré de taille $n \times n$ telle que $A_{ij} = 1$ s'il existe une arête allant de i vers j et $A_{ij} = 0$ sinon. Écrire une fonction `adjmat(graph)` qui prend en entrée un graphe `G` et renvoie la matrice d'adjacence associée (sous forme de tableau NumPy).

d) On considère un utilisateur qui navigue entre n pages web, numérotées de 0 à $n-1$, aléatoirement de la façon suivante :

- Si l'utilisateur se trouve sur une page i qui contient m liens vers des pages distinctes et différentes de i, il choisit uniformément au hasard un lien parmi les m possibles et y va.
- S'il se trouve sur une page qui ne pointe vers aucune autre page, alors il choisit uniformément au hasard une page parmi toutes les $n-1$ pages et y va.

Écrire une fonction `probmat(graph)` qui a un graphe renvoie la matrice des probabilités (sous forme de tableau NumPy) qui a en position i, j la probabilité que l'utilisateur saute à la page j en partant de la page i.

e) Soit M une matrice obtenue par la fonction `probmat`. Pour $k > 0$, on note M^k le produit matriciel k fois de M avec elle même. Montrer par récurrence sur k que pour tout $0 \leq i \leq n-1$ et $0 \leq j \leq n-1$, $(M^k)_{ij}$ correspond à la probabilité pour un utilisateur partant de la page i de se retrouver à la page j après k étapes.

f) Soit $\boldsymbol{v}$ le vecteur de taille n définit par $\boldsymbol{v} = (1/n, 1/n..., 1/n)$. Montrer que pour tout $0 \leq i \leq n-1$, la i-ème composante du vecteur $\boldsymbol{v}M$ correspond à la probabilité qu'a un utilisateur ayant commencé sur une page uniformément aléatoire se retrouve sur la page i après avoir suivi un lien. Que peut-on en déduire de la i-ème compostante du vecteur $\boldsymbol{v}M^k$?

g) Écrire une fonction `pagerank(graph,k)` qui prend en entrée un graphe ainsi qu'un entier `k` et retourne le vecteur $\boldsymbol{v}M^k$.

h) [bonus] On considère maintenant qu'à chaque étape de navigation, avec probabilité $p \in [0,1]$, l'utilisateur décide de retourner vers une page aléatoire. Écrire une fonction `pagerank2(G,k,p)` qui retourne le vecteur des probabilités qu'un utilisateur se retrouve sur chaque page après k étape de cette nouvelle navigation.

Exercice 3 : Génération aléatoire de vecteurs unitaires

L'objectif de cet exercice est de trouver une méthode efficace pour générer aléatoirement des vecteurs unitaires dans $\mathbb{R}^n$ selon une loi uniforme. Nous allons commencer par le cas $n = 2$ où l'on peut représenter un vecteur réel par un nombre complexe.

a) On considère la stratégie suivante pour générer aléatoirement un vecteur unitaire dans $\mathbb{R}^2$:

1. Générer x aléatoirement selon la loi uniforme sur $[-1,1]$;
2. Générer y aléatoirement selon la loi uniforme sur $[-1,1]$;
3. Renvoyer le complexe unitaire $z = \frac{x}{\sqrt{x^2+y^2}} + \frac{y}{\sqrt{x^2+y^2}}i$.

Écrire une fonction `generer_complexe` qui prend en argument un entier positif N et qui renvoie un tableau NumPy de taille N où chaque élément un complexe généré par la stratégie ci-dessus.

Indication : On pourra utiliser les fonctions `numpy.random.uniform` et `numpy.abs`. **Attention :** il est beaucoup plus efficace de générer directement N variables aléatoires en utilisant l'argument `size` que de les générer une par une avec une boucle `for`.

b) Pour $n = 2$, une manière de vérifier si la distribution des vecteurs est uniforme est de regarder la distribution des angles/arguments des nombres complexes (c'est-à-dire $\arg z$) : celle-ci doit être uniforme. Utiliser la fonction `generer_complex` avec $N = 10^6$ et afficher la distribution des angles/arguments. Est-ce que la stratégie présentée à la question précédente permet-elle de générer des vecteurs unitaires de manière uniforme ?

Indication : On pourra utiliser les fonctions `numpy.angle` et `matplotlib.pyplot.hist`.

c) On proprose la modification suivante de la stratégie précédente :

1. Générer x aléatoirement selon la loi uniforme sur $[-1,1]$;
2. Générer y aléatoirement selon la loi uniforme sur $[-1,1]$;
3. Si $x^2+y^2 \leq 1$, renvoyer le complexe unitaire $z = \frac{x}{\sqrt{x^2+y^2}} + \frac{y}{\sqrt{x^2+y^2}}i$ (sinon ne rien renvoyer du tout).

Écrire une fonction `generer_complexe_monte_carlo` qui prend en argument un entier positif N correspondant au nombre de vecteurs candidats et qui renvoie un tableau NumPy de taille $n \leq N$ où chaque élément un complexe généré par la stratégie ci-dessus. **Attention :** n est aléatoire et ne peut donc pas être déterminé à l'avance.

Reprendre la question précédente et afficher la distribution des angles/arguments des complexes générés par cette stratégie. Cette stratégie permet-elle de générer des vecteurs unitaires de manière uniforme ?

d) On se demande maintenant combien de complexes candidats il faut générer en moyenne pour en accepter K, ce qui est équivalent à déterminer combien de complexes candidats sont en moyenne acceptés parmi les N complexes candidats (les ratios sont inversés). La loi des grands nombres permet d'affirmer que le ratio converge vers la probabilité qu'un vecteur candidat soit accepté. Cette probabilité est égale au ratio de la surface d'inclusion π (le cercle unitaire) sur la surface totale 4 (le carré $[-1,1]^2$), c'est-à-dire $\frac{\pi}{4}$. Comparer le ratio $\frac{n}{N}$ à $\frac{\pi}{4}$.

e) On se place maintenant dans le cas général $n \geq 2$. La stratégie considérée est la même que celle à la question **c)** :

1. Générer un vecteur $\boldsymbol{x} = (x_1, \ldots, x_n)$ où chaque x_k est généré aléatoirement et de manière indépendante selon la loi uniforme sur $[-1, 1]$;
2. Si $\|\boldsymbol{x}\|_2 \leq 1$, renvoyer le vecteur unitaire $\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$.

Le volume de la boule unitaire dans $\mathbb{R}^n$ est donné par :

$$V_n = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2} + 1)}$$

et le volume du cube $[-1, 1]^n$ est 2^n. Afficher sur un graphique la probabilité qu'un vecteur soit accepté à l'étape 2 de cette stratégie pour $n \in \{2, \ldots, 20\}$. Pensez-vous que cette stratégie est efficace pour de grandes valeurs de n ?

Indication : On pourra utiliser `scipy.special.gamma` pour la fonction Γ et la fonction `matplotlib.pyplot.scatter` pour afficher graphiquement les valeurs d'une suite avec une échelle adaptée.

f) Il est possible de montrer que la stratégie suivante permet de générer des vecteurs aléatoires sur $\mathbb{R}^n$:

1. Générer un vecteur $\boldsymbol{x} = (x_1, \ldots, x_n)$ où chaque x_k est généré aléatoirement et de manière indépendante selon la loi normale centrée réduite : $x_k \sim \mathcal{N}(0, 1)$;
2. Renvoyer le vecteur unitaire $\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$.

Vérifier pour $n = 2$ que cette stratégie génère bien aléatoirement des vecteurs unitaires selon une loi uniforme en affichant la distribution des angles (en représentant les vecteurs comme des nombres complexes) pour $N = 10^6$ vecteurs.

Indication : On pourra utiliser la fonction `numpy.random.randn`.

g) [bonus] Interpréter le graphique obtenu à la question **b)**.