BE : Variations autour de la reconnaissance des chiffres manuscrits

Par groupe de 4 étudiants, vous devez rédiger un rapport dans lequel vous répondrez aux questions du sujet. La qualité de rédaction ainsi que tous les compléments que vous apporterez (réflexions, analyses des réponses, tests sur d'autres données, etc...) seront fortement pris en compte. Toutes les fonctions programmées le seront en Python et seront abondamment commentées. Le rendu du projet se fera sous la forme d'un fichier .zip ou .rar contenant le rapport ainsi que l'ensemble des programmes. Attention certaines questions demandent à optimiser des paramètres lors de calcul un peu long (environ 5min à 10min), il est conseillé de partager ces calculs d'optimisations sur l'ensemble des ordinateurs du groupes afin de gagner du temps.

Dans tout ce BE nous utiliserons l'exemple vu en TP (cf. TP4) de la reconnaissance des chiffres manuscrits à partir de la base de données MNIST. Nous essaierons d'améliorer quelque peu le taux de reconnaissance. Dans la partie 1, nous verrons une méthode pour résoudre ce problème avec la factorisation de Cholesky en modifiant légèrement le problème et en le faisant dépendre d'un paramètre. En optimisant ce paramètre nous pourrons alors augmenter quelque peu le taux de réussite dans le cas de la reconnaissance d'un chiffre. Dans la partie 2, nous verrons une approche « géométrique » sur la reconnaissance des chiffres manuscrits en utilisant l'analyse procustéenne. Enfin dans la troisième partie nous couplerons les deux approches précédentes afin d'augmenter le taux de réussite. Nous verrons au second semestre comment grandement augmenter le taux de réussite de reconnaissance avec des méthodes plus élaborées.

Partie 1 : Inverser le non-inversible

Pour tester nos algorithmes nous allons utiliser la base de données classique MNIST (Mixed National Institute of Standards and Technology).

On rappelle que cette base de données contient un jeu de 60 000 images de chiffres écrits à la main et 10 000 images tests afin de vérifier les performances des algorithmes. Chacun de ces deux jeux de données est fourni avec des « labels/étiquettes » afin de vérifier les résultats obtenus. Chacune de ces images de chiffres manuscrits est une image en nuance de gris de taille 28×28.

```
train_data = np.loadtxt("mnist_train.csv",delimiter=",") #données d'entrainement
test_data = np.loadtxt("mnist_test.csv",delimiter=",")   #données de test
```

On rappelle également que la première colonne de chacun des deux fichiers de données contient les « étiquettes » qui sont les chiffres représentés sur l'image. Les autres colonnes contiennent une image de chiffre qui est implémenter sous la forme d'un vecteur ligne : opération de vectorisation d'un vecteur (les lignes d'une matrice sont mises côtés à côtés).

On rappellera également dans la suite les notations pour un problème de classification « linéaire » binaire. Supposons donné un ensemble fini E partitionné en deux sous-ensembles disjoints de points de $\mathbb{R}^n$:

$$E_1 = \{u_i \in \mathbb{R}^n \,|\, i \in [\![1,p]\!]\} \quad \text{et} \quad E_2 = \{v_j \in \mathbb{R}^n \,|\, j \in [\![1,q]\!]\}$$

où p et q sont des entiers non nuls. Ces données forment ce que l'on appelle les données d'entraînement et on cherche une fonction $f : \mathbb{R}^n \to \mathbb{R}$ de la forme (fonction affine) :

$$f(x) = w^T x + b$$

avec $w \in \mathbb{R}^n$ et $b \in \mathbb{R}$ telle que :

$$f(u_i) = 1 \quad \forall i \in [\![1,p]\!] \qquad \text{et} \qquad f(v_j) = -1 \quad \forall i \in [\![1,q]\!]$$

Comme il n'est pas clair qu'une telle fonction existe nous allons rechercher f qui minimise la quantité suivante :

$$\sum_{i=1}^{p} \left(f\left(u_i\right) - 1\right)^2 + \sum_{j=1}^{q} \left(f\left(v_j\right) + 1\right)^2$$

Cette quantité quantifie l'erreur de prédiction de f sur les données d'entraînement.

1. Rappeler que ce problème se réécrit sous la forme matricielle suivante :

$$\min \sum_{i=1}^{p} \left(f\left(u_i\right)-1\right)^2+\sum_{j=1}^{q} \left(f\left(v_j\right)+1\right)^2=\min_{x\in\mathbb{R}^{n+1}} \|Ax-y\|_2^2$$

avec A et y que vous préciserez.

2. En notant :

$$\phi(x):=\|Ax-y\|_2^2$$

Justifier que les points critiques de ϕ vérifient l'équation dite équation normale suivante :

$$A^TAx=A^Ty$$

3. Sous Python, en utilisant le cas de l'apprentissage de la reconnaissance du chiffre 0, calculer les valeurs propres approchées de la matrice A^TA en utilisant l'algorithme d'itération QR, puis donner une approximation du rang de la matrice A. Justifier que A^TA ne peut être une matrice inversible. On pourra utiliser la fonction Python *np.linalg.qr*($\bullet$) qui renvoie la décomposition QR d'une matrice.
4. Comparer la réponse obtenue à la question précédente avec l'estimation du rang en utilisant le calcul des valeurs singulières par la commande *np.linalg.svd*($\bullet$). Que pouvez-vous en conclure ?
5. Dans le TP4, nous avons vu qu'il est possible de résoudre le problème précédent et de donner une solution en utilisant le pseudo-inverse de A :

$$x^*:=A^{\dagger}b$$

Or nous allons choisir une autre stratégie ici : nous allons modifier la matrice A^TA afin de la rendre inversible. Nous allons poser la matrice suivante :

$$A_\epsilon:=A^TA+\epsilon I_{785}$$

où ϵ est un réel strictement positif et I_{785} est la matrice identité de taille 785×785.

(a) En utilisant encore une fois le cas de l'apprentissage de la reconnaissance du chiffre 0, calculer les valeurs propres approchées de la matrice A_ϵ en fonction de ϵ. Justifier par le calcul Python que cette matrice est symétrique définie positive.

(b) Écrire sous Python une fonction **resChol(nombredetection,epsilon)** qui dépend d'un entier *nombredetection* $\in [\![0,9]\!]$ correspondant au nombre à détecter et *epsilon* au ϵ de la question précédente. Cette fonction :

i. calculera la solution :

$$sol_\epsilon:=A_\epsilon^{-1}A^Ty$$

en utilisant la décomposition de Cholesky de A_ϵ et la résolution d'un système triangulaire inférieur puis enfin d'un système triangulaire supérieur à préciser.

ii. renverra le taux de réussite de reconnaissance du chiffre *nombredetection* :

$$T_r:=\frac{N_{vp}+N_{vn}}{10000}$$

ainsi que la matrice de confusion M_{conf} associée sur les 10 000 données test (test_data) :

$$M_{conf}:=\begin{pmatrix} N_{vp} & N_{fn}\\ N_{fp} & N_{vn}\end{pmatrix}$$

où N_{vp} le nombre de vrais positifs, N_{fn} le nombre de faux négatifs, N_{fp} le nombre de faux positifs et N_{vn} le nombre de vrais négatifs.

(c) Calculer le taux de réussite ainsi que la matrice de confusion associée pour chaque chiffre *nombredetection* $\in [\![0,9]\!]$ pour $\epsilon=1$.

(d) En faisant varier ϵ sur l'intervalle $[10^{-10},10^9]$ et $\epsilon\neq 0$, tracer le taux de réussite en fonction de ϵ en utilisant la fonction $resChol(0,\epsilon)$. On pourra découper sous python cet intervalle en 50 parties pour commencer puis cibler des intervalles d'études afin de trouver empiriquement un ϵ qui maximise le taux de réussite.

(e) Pour chaque chiffre $i\in [\![0,9]\!]$, trouver ϵ_i qui maximise la détection du chiffre i. Il est conseillé de vous partager le travail sur plusieurs ordinateurs. Donner pour chaque chiffre le meilleur taux de réussite obtenu ainsi que la matrice de confusion associée. Enregistrer le vecteur solution pour chaque chiffre avec la commande **np.save**.

6. En utilisant les résultats des fonctions $resChol(i, \epsilon_i)$ pour $i \in [\![0, 9]\!]$ trouver à la question précédente, rédiger un programme de détection des chiffres. Vous définirez la fonction **fglobale(x,SOL)** où x est une image vectorisée d'un chiffre manuscrit et comme sortie un vecteur de taille 10 dont les composantes sont données par les fonctions $f_i = resChol(i, \epsilon_i)$ calculées précédemment et SOL est la matrice de taille 785×10 contenant les vecteurs colonnes i de la forme :
$$sol_i := \begin{pmatrix} w_i \\ b_i \end{pmatrix}$$
avec w_i et b_i les vecteurs et scalaires obtenus lors du calcul de f_i. On prendra comme prédiction l'indice i de la composante f_i de **fglobale(x,SOL)** la plus grande. Donner alors son taux de réussite sur les données test (test_data). Comparer avec le taux de réussite trouver en TP avec le pseudo-inverse. Que concluez-vous ?
7. (Bonus) Rédiger une interface Tkinter qui permet prédire un chiffre manuscrit sur des données tests en utilisant la fonction précédente. Votre interface devra comporter l'affichage de l'image d'un chiffre manuscrit des données tests et le résultat de la prédiction de votre algorithme produit à la question précédente.
8. Théorisons à présent pourquoi le remplacement de la matrice $A^T A$ par $A^T A + \epsilon I_n$ fonctionne. Nous avons vu que la résolution de :
$$\min_{x \in \mathbb{R}^{n+1}} \|Ax - y\|_2^2$$
revient à rechercher les points critiques de la fonction ϕ définie par :
$$\phi(x) := \|Ax - y\|_2^2 = 2\left(x^T A^T A x - x^T A^T b\right)$$
i.e. que les points critiques de ϕ vérifie l'équation normale :
$$A^T A x = A^T b$$
   (a) Montrer que le fait de remplacer $A^T A$ par $A^T A + \epsilon I_n$ revient à rechercher les points critiques de la fonction :
$$\phi(x) := \|Ax - y\|_2^2 + \epsilon \|x\|_2^2$$
   (b) En déduire l'influence du facteur ϵ sur la forme de la solution du problème :
$$\min_{x \in \mathbb{R}^{n+1}} \|Ax - y\|_2^2 + \epsilon \|x\|_2^2$$

Partie 2 : L'approche procustéenne

Nous allons aborder dans cette partie le problème de la reconnaissance des chiffres manuscrits du point de vue de l'analyse procustéenne. L'idée est ici de comparer l'image (« vectorisée ») d'un chiffre manuscrit inconnu (i.e. une ligne de test_data, sans la colonne label bien sûr...) avec les images « moyennes » des chiffres de 0 à 9 sur les données d'entraînements.

1. Sur les données train_data[:,1 :], les commandes suivantes :

```
nombredetection=0
valeurs=train_data[:,0]
indiceu=np.where(valeurs==nombredetection)
u=train_data[:,1:][indiceu]
```

   donne une matrice u qui contient toutes les images des chiffres 0 contenu dans les données d'entraînement. En utilisant la commande de Numpy : np.mean() de manière appropriée crée un vecteur de taille $(1, 784)$ que l'on appellera *zeromoyen*, qui est la moyenne de tous les chiffres 0 présent dans train_data. Après avoir redimensionné ce vecteur et l'avoir converti en « uint8 », afficher l'image du 0 moyen.
2. En généralisant la question précédente, créer une fonction **chiffremoy(train_data)** qui renvoie une matrice de taille 10×784 où chaque ligne i représente le vecteur du chiffre i moyen. Utiliser cette fonction pour afficher les images correspondantes aux chiffres moyens.
3. Soient $A, B \in \mathcal{M}_{mn}(\mathbb{R})$ deux matrices. On rappelle que l'analyse procustéenne (linéaire) consiste à rechercher Φ une transformation affine i.e. s'écrivant matriciellement :
$$\Phi(A) = \lambda X A + tu$$
   où les inconnues sont :
   — $\lambda \in \mathbb{R}$ le rapport d'homothétie,

— $X \in \mathbb{O}_m$, une transformation orthogonale,

— $t \in \mathbb{R}^m$ est un vecteur de translation (vu comme vecteur colonne) et u le vecteur (vecteur ligne) de $\mathbb{R}^n$: $u = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}$

telle que Φ minimise la quantité suivante :

$$\min_{\Phi \text{ transformation affine}} \|B - \Phi(A)\|_F^2$$

On rappelle également que la solution à ce problème est donnée par :

$$\boxed{X = V_G U_G^T} \qquad \boxed{t = b_G - \lambda X a_G} \quad \text{et} \quad \boxed{\lambda = \frac{trace\,(\Sigma_G)}{\|A_G\|_F^2}}$$

où U_G et V_G sont données par une décomposition en valeurs singulières (SVD) de $A_G B_G^T = U_G \Sigma_G V_G^T$ avec :

$$A_G = A - a_G u \qquad \text{et} \quad B_G = B - b_G u$$

et :

$$b_G := \frac{1}{n}\sum_{j=1}^{n} b_j \quad \text{et} \quad a_G := \frac{1}{n}\sum_{j=1}^{n} a_j$$

avec les a_i et b_i les vecteurs colonnes de A et B respectivement.

Écrire une fonction $Procuste(A, B)$ qui prend en entrée deux matrices de même taille et qui renvoie en sortie le tuple :

$$\left(\lambda, X, t, \|B - \Phi(A)\|_F^2\right)$$

On appellera la quantité : $\|B - \Phi(A)\|_F^2$: **l'erreur de transformation**.

Attention en algèbre, la convention est de prendre les données sous forme de vecteurs colonnes. Or dans base de données MNIST (comme dans le cas de la base de données Iris du TP5) les données liées à un objet/individu sont sous forme de vecteurs lignes. Nous choisirons comme cela a été fait en TP5 cette dernière convention dans la suite de cette consigne : les données liées à un objet/individu (ici image) sont sous forme de lignes en entrée de la fonction Procuste(A,B). Il faut donc les convertir en vecteurs colonnes dans la fonction si vous souhaiter appliquer les résultats du rappels directement.

4. En notant CM :

```
CM=chiffremoy(train_data)
```

la matrice incarnant les chiffres moyens. Implémenter une fonction **comparaison(x, CM)** où x est une image de test_data i.e x est une ligne de la forme :

```
x=train_data[i,1:].reshape((784,1))
```

avec i un entier compris entre 0 et 9999. Cette fonction renverra un tuple : **veccomp**, **resultat** où :

- **veccomp** est un array de taille 10 ayant pour composante j l'erreur de transformation de l'analyse procustéenne entre, avec les notations précédentes du rappel sur l'analyse procustéenne : $A = CM[i, :].reshape((1, 784))$ et $B = x.T$ (**attention** : on rappelle que les données d'entrées sont sous forme de vecteurs lignes... il faut donc les convertir en colonne dans la fonction Procuste(•)),
- **resultat** est un entier. C'est l'indice de la composante maximale du vecteur **veccomp**. Cette valeur est le résultat de la prédiction sur le chiffre manuscrit correspondant à x par l'analyse procustéenne.

5. Réaliser une fonction qui donne le taux de reconnaissance par cette méthode sur l'ensemble des chiffres manuscrits des données tests. Comparer ce taux de réussite avec celui de la partie 1. Qui gagne ?

Partie 3 : Le mélange

Dans cette partie nous allons nous servir des deux algorithmes développés dans les parties précédentes pour réaliser un algorithme de prédiction que l'on espère plus précis.

1. En reprenant les notations précédentes, rédiger une fonction de reconnaissance des chiffres manuscrits qui pour une donnée test sous la forme :

```
x=train_data[i,1:].reshape((784,1))
```

renvoie le vecteur **resultat** de taille 10 défini de la façon suivante :

```
v=coeff*(1/comparaison(test_data[i,1:],CM)[0])
rep=fglobale(test_data[i,1:].reshape((784,1)),SOL).reshape(np.shape(v))
resultat=v+rep
```

où $coeff$ est un réel (flottant) strictement positif. La prédiction sera comme dans les partie précédentes l'indice de la composante maximale du vecteur **resultat** précédent.

2. Calculer le taux de réussite de cette algorithme en fonction du $coeff$ choisi. Trouver une valeur qui optimise ce taux de réussite.

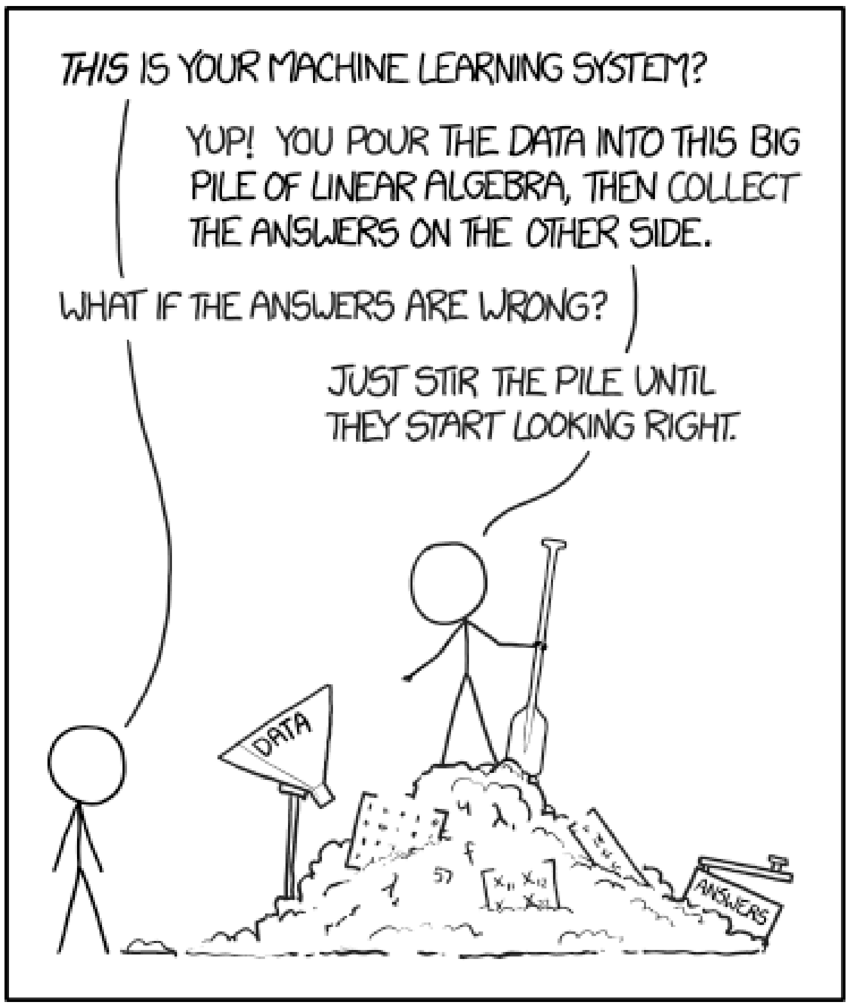

Figure 1 – https ://xkcd.com/1838/