Factorisation Matricielle Non-Négative

Pour la reconnaissance de chiffres manuscrits

Réalisé par :

Lydia KHRIS

Kevin TATIBOUET

Pranavan RAJENDIRAN

Louise DAUDIN

Piraveena CHIVANESAN

Encadré par : Mr Basarab MATEI

Année universitaire : 2020-2021

Sommaire :

INTRODUCTION :

Nous avons réalisé un projet de factorisation matricielle non-négative sur la reconnaissance de chiffres manuscrits en python.

Le but de ce projet était d'utiliser les différents algorithmes de traitement de données sur le MNIST DataSet, à savoir PCA, NMF, ProjectiveNMF, SPARSITY.

Le traitement, l'analyse et la factorisation de ces données nous ont permis d'obtenir une représentation de rang inférieur de matrice avec des éléments positif.

Vous trouverez ci-joints les liens des sources que nous avons utilisé pour répondre à certaines questions.

1) https://github.com/canerturkmen/nmflib/blob/master/projective.py (nmfprojective)

2) https://www.jmlr.org/papers/volume5/hoyer04a/hoyer04a.pdf (sparsityh)

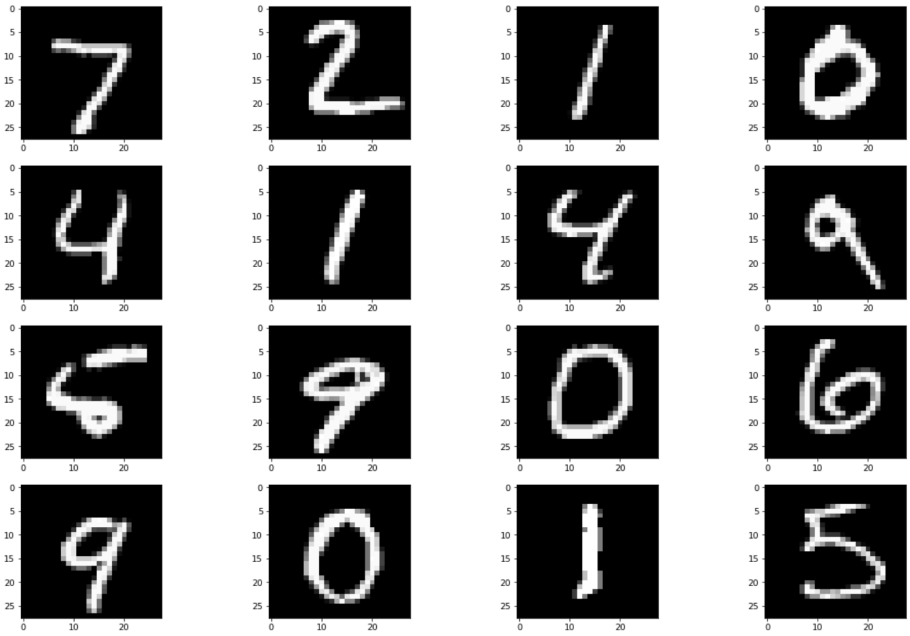
3) https://www.kaggle.com/gainknowledge/mnist-scikit-learn-tutorial (showchiffre)

2) VISUALISATION DES CHIFFRES MANUSCRITS INITIALE

-Nous avons utilisé Le **mnist_test.csv un des deux fichiers** de données MNIST au format CSV (Mnist_train.csv et mnist_test.csv)

Le **mnist_test.csv** contient 10 000 exemples de tests et étiquettes. Chaque ligne se compose de 785 valeurs : la première valeur est l'étiquette (un nombre de 0 à 9) et les 784 (28*28) valeurs restantes sont les valeurs de pixel (un nombre de 0 à 255).

- Afin de visualiser par exemple les 12 premiers chiffres de notre DataSet, nous avons implémenté une fonction showchiffre qui prend en arguments le DataSet, le nombre de lignes et de colonnes ainsi que le titre de l'image renvoyée et renvoie le plot qui contient les 6 premiers chiffres manuscrits de notre DataSet.



APPLICATION DE PCA (Analyse en Composantes Principales):

Les questions a et b de cette partie nous les avons faites de deux manières différentes.

1) Calcul de la moyenne manuellement pour ensuite l'utilise pour la création de la matrice A.

2) Utilisation de **StandardScaler** de la bibliothèque sklearn. (Supprime la moyenne et la remplace par la variance unitaire, utilise fit_ transform qui calcule les paramètres de l'échantillon (mean), puis les transformer et renvoie un array).

Questions c, d et e:

PCA est une des méthodes les plus importantes de l'apprentissage non supervisé, le principe est de réduire la complexité superflue d'un dataset en projetant ses données dans un espace de plus petites dimension (moins de variables). On l'utilise pour accélérer l'apprentissage de la machine et pour lutter contre le fléau dimensionnel.

Le principe est de projeter nos données sur des axes appelés composantes principales en cherchant à minimiser la distance entre nos points et leurs projections ainsi on réduit la dimension de notre dataset et on conserve au max la variance des données.

(PCA fonctionnement : préciser le nombre de dimensions sur lesquelles on souhaite projeter nos données puis on utilise fit_transform .)

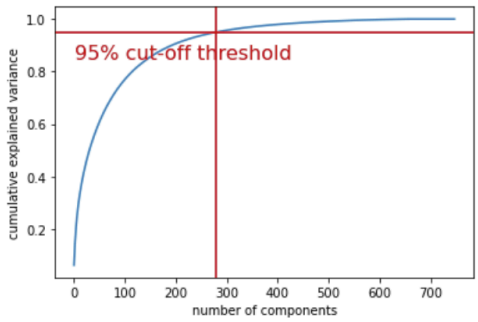
Compression des données :

Pour avoir le nombre de component qui permet de garder 95% de la variance avec x=np.argmax(np.cumsum(pca.explained_variance*ratio*) > 0.95) qui nous renvoie 279.

```
       [0., 0., 0., ..., 0., 0., 0.]])
```

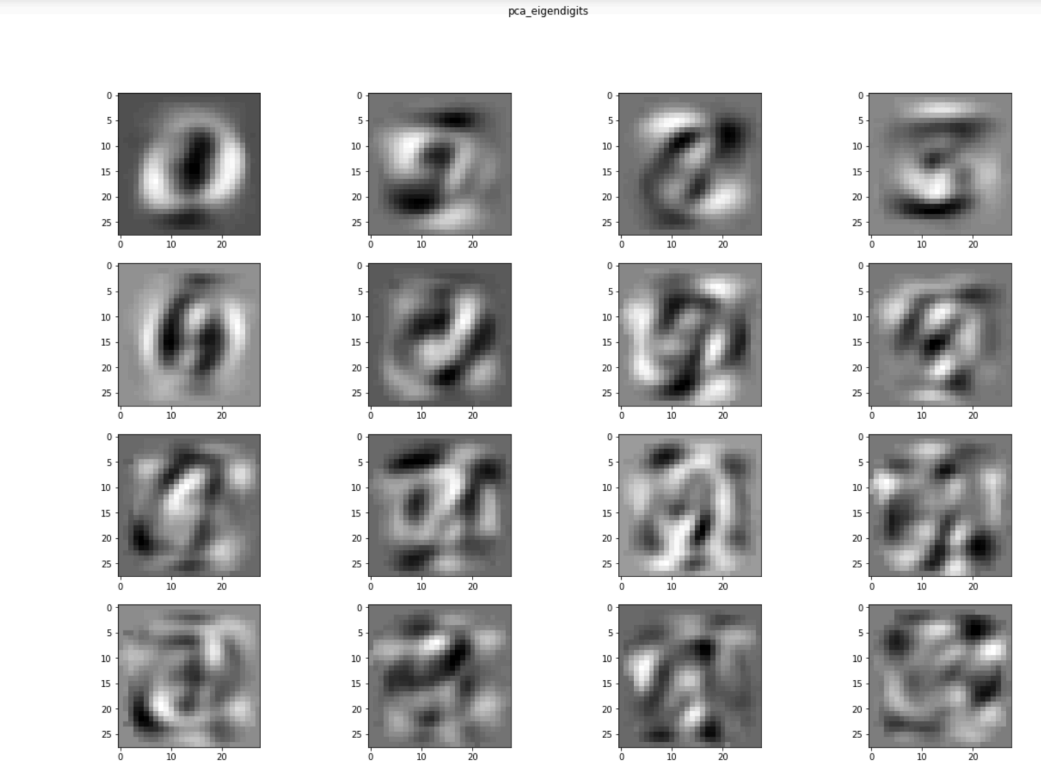
Entrée [23]:
```
#3.c, 3.d, 3.e
pca=PCA(n_components=748)
pca.fit_transform(A)
pca_eigendigit=pca.components_

#show_chiffre(pca_eigendigit,4,4,"pca_eigendigits")

# pour la courbe il faut changer n_components =748

x=np.argmax(np.cumsum(pca.explained_variance_ratio_) > 0.95)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
plt.axhline(y=0.95, color='r', linestyle='-')
plt.axvline(x=x, color='r', linestyle='-')
plt.text(0.5, 0.85, '95% cut-off threshold', color = 'red', fontsize=16)
x
```

Out[23]: 279



Une fois la valeur trouvée, on réapplique PCA avec le n_components =279

Puis on affiche avec showchiffre le résultat suivant :



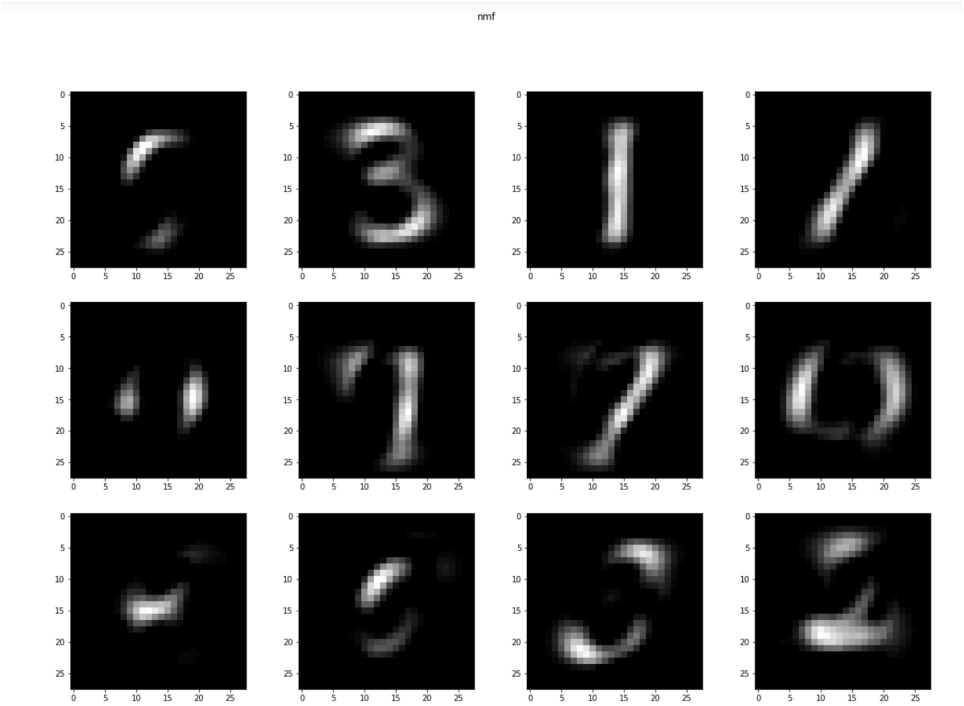
APPLICATION DE NMF (Factorisation de Matrice non-Négative)

Factorisation matricielle non négative, est une technique permettant d'obtenir une représentation de rang inférieur de matrices avec des éléments non négatifs ou positifs. Par exemple les images de notre digits dataset est un ensemble de matrice de nombre entier positif représentant des intensités de pixels.

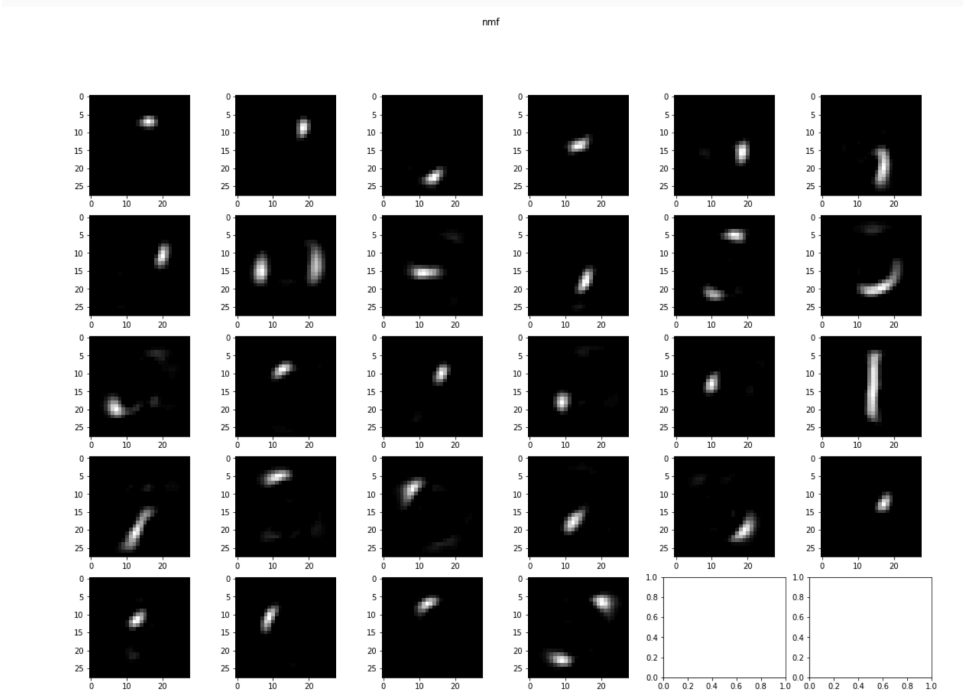
Cette technique sert à Trouver deux matrices non négatives (W, H) dont le produit se rapproche de la matrice X non négative. Cette factorisation peut être utilisée par exemple pour la réduction de la dimensionnalité, la séparation de source ou l'extraction des données. Enfin nous obtenons un ensemble d'images imaginaires qui nous permettra de reconstruire toutes nos données de la base.

Après l'application de NMF sur nos données initiales nous avons obtenus ces résultats :

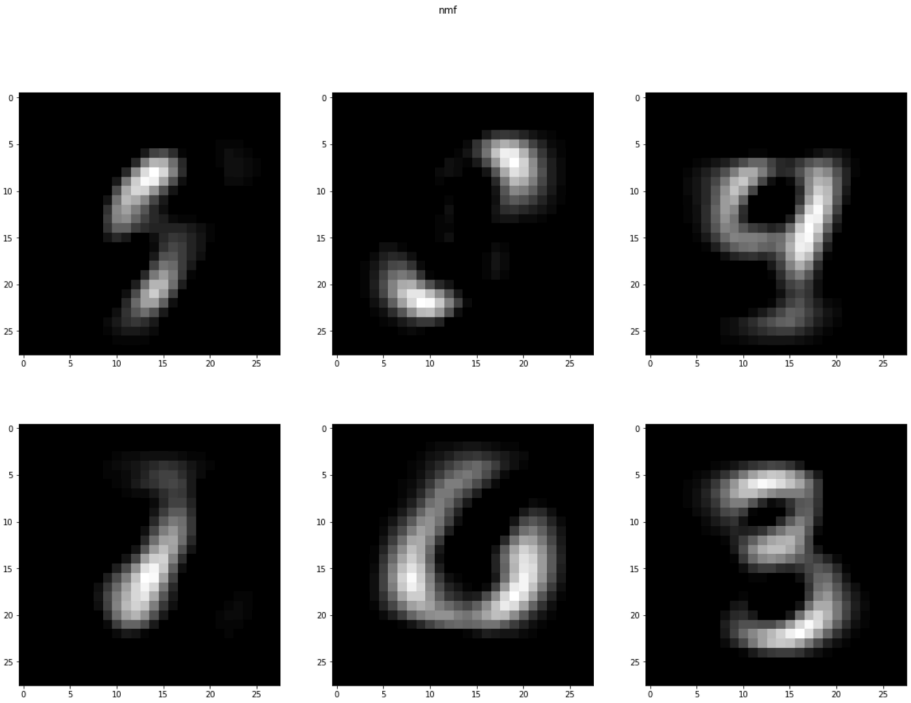
N_components = 12



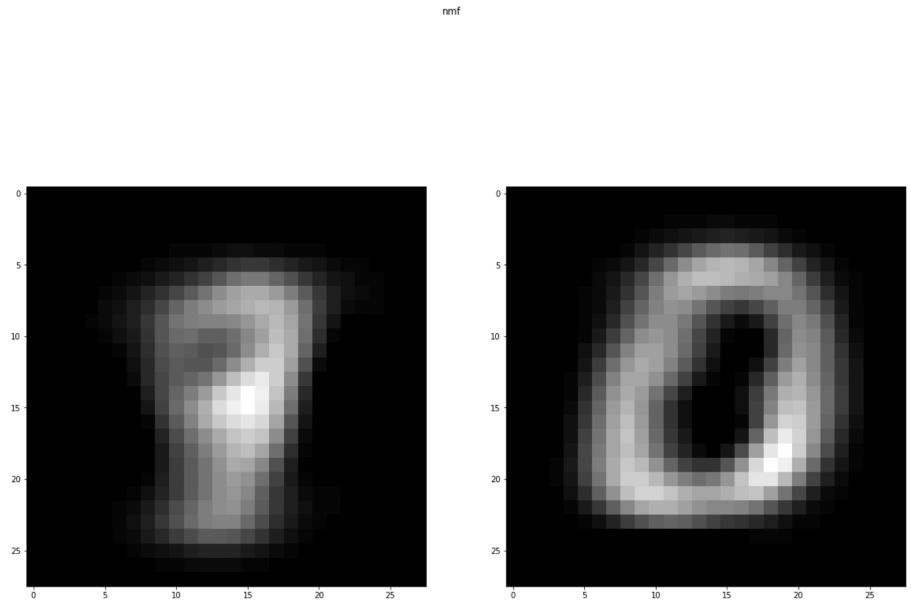
N_components = 28



N_components = 6



N_components = 2



APPLICATION DE PROJECTIVE NMF :

On fait une projection de toutes nos données et on application NMF sur ces données projetées.

En se basant sur notre cours nous avons suivis cette technique de ProjNMF :

NMF : $X \approx WH$

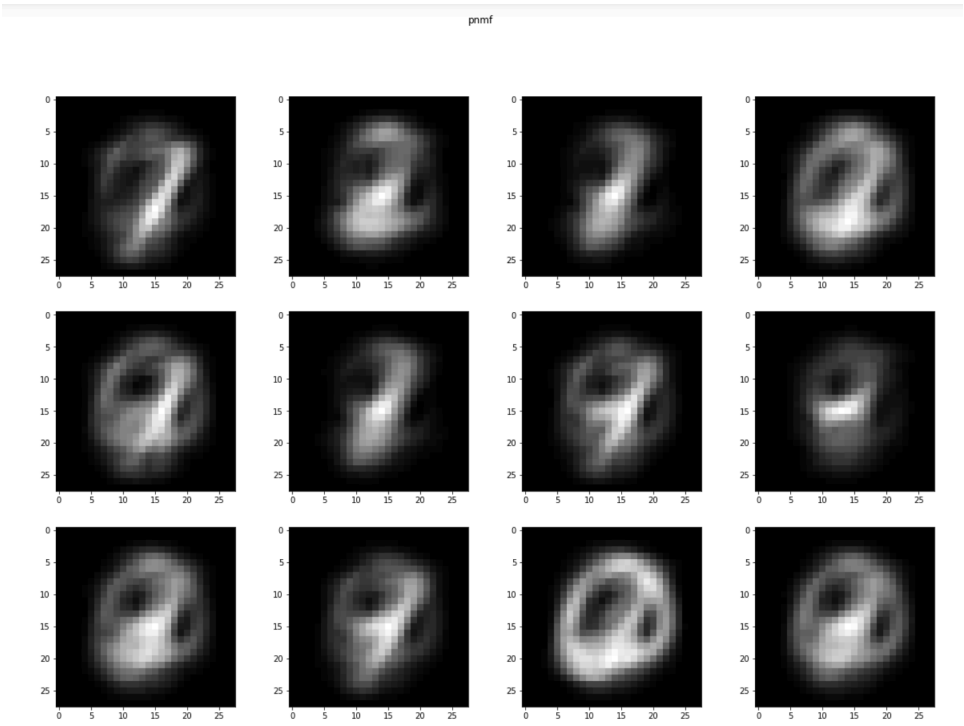
Projective NMF : $X \approx WW^TX$

Avec des manipulations mathématiques nous avons :

$WH \approx WW^T X$ alors $H \approx W^TX$

Nous allons construire Projective NMF en se basant sur ces manipulations ci-dessus ce qui nous mène à avoir une approximation de $X \approx WW^TX$

Cela est le résultat de l'application de notre fonction ProjNMF (W, X).



SPARSITY :

Fait référence à un schéma de représentation où seules quelques unités (sur une grande population) sont effectivement utilisées pour représenter des vecteurs de données typiques, en effet, cela implique que la plupart des unités prennent des valeurs proches de zéro alors que seules quelques-unes prennent des valeurs significativement non nulles.

Après l'exécution notre fonction sparsity sur l'ensemble de nos données pour calculer le taux de d'éléments nul dans notre matrice issue de NMF obtenue.

```
def sparsity(A):
    return (1.0 - float(np.count_nonzero(A)) / A.size)
```

Les résultats de l'application de sparsity sur PCA, NMF, PNMF :

```
PCA: 0.07491679467485923
NMF: 0.6994047619047619
PNMF: 0.14795918367346939
```

SPARSITYH :

SparsityH (pour sparsity de Hoyer) est un nombre réel compris entre 0 et 1. Il est égal à 1 si et seulement si la matrice mise en paramètre de la fonction contient un seul composant différent de zéro et est égal à 0 si tous les composants de cette matrice sont égaux.

De par nos recherches nous avons trouvé la fonction de parcimonie en Matlab que nous avons par la suite implémenté en python.

```
function [S A] = sparsityh(X)
for i = 1:size(X,2)
        A(i) = (sqrt(size(X,1)) - (sum(abs(X(:,i)))/sqrt(sum(X(:,i).^2))))/(sqrt(size(X,1)) - 1);
End
S = mean(A);
```

Les résultats de l'application de sparsityH sur PCA, NMF, PNMF :

```
PCA-H: -0.0007932397470768679
NMF-H: -0.2738218306843233
PNMF-H: 8.39759404445909e-05
```

CONCLUSION :

Le but de ce projet est d'appliquer les différentes techniques de PCA, NMF à l'analyse des images de Digits dataset et de visualiser les divers résultats.

Nous avons pu approfondir certaines de nos connaissances comme PCA et mettre en pratique la théorie du cours et cela pour bien comprendre la fonctionnalité des différentes techniques de reconnaissance des images.