POLYTECH PARIS-SUD

Et4 INFO

# Rapport de MachineLearning

Projet de reconnaissance d'images


**Xiaoxuan HEI**

**Jiangnan HUANG**

Groupe 1

25 mars 2019

# Introduction

Le but de ce projet est de mettre en place et d'évaluer un système de reconnaissance automatique d'image en Python.

# Partie1 : Classifieur à distance minimum

Question : Implémentation du classifieur à distance minimum (DMIN) : chaque classe est représentée par la moyenne des vecteurs d'apprentissage de cette classe. Ce classifieur doit être réalisée en Python en utilisant uniquement les librairies Numpy et Matplotlib. Vous calculerez la performance sur l'ensemble de développement grâce au taux d'erreur, défini comme le nombre d'exemples mal classés divisé par le nombre total d'exemples.

## Implémentation :

```
for i in range(10):
    z[i] = x[y==i]
```

Séparer les éléments et les stocker en fonction de la classe

```
for i in range(10):
    m[i] = np.mean(z[i],axis=0)
```

On calcule les représentants de chaque classe et puis calcule les distances entre les représentants et les images de développement pour classifier toutes les images. On compare le résultat avec les étiquettes réelles pour calculer le taux d'erreur.

## Résultat obtenu :

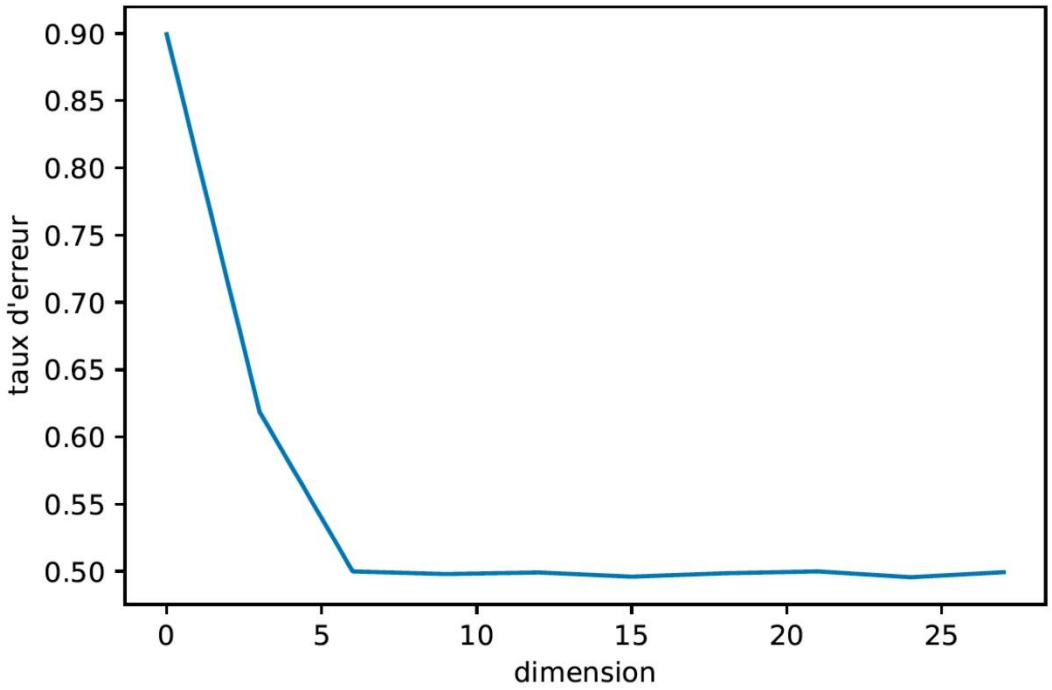
Après l'exécution, on a constaté que le taux d'erreur est 0.3242.
Le temps de l'apprentissage est de 0.0219 sec.
Le temps de la classification est de 0.1637 sec.

# Partie2 : Analyse en composantes principales (PCA)

Question : Réduction de la dimension des vecteurs : l'analyse en composantes principales (ACP) permet de réduire la dimension des vecteurs d'images de 784 points à un vecteur de paramètres de plus petite taille (voir votre cours ou l'annexe ci-dessous pour la théorie, en pratique vous n'avez pas à l'implémenter vous-même car vous utiliserez la librairie Scikit-Learn sklearn.decomposition.PCA). Vous testerez l'impact de la réduction de dimension sur le classifieur précédent (PCA+DMIN) en fonction du choix de la dimension de l'ACP (par exemple sur une dizaine de valeurs représentatives).

## Implémentation :

Nous avons créé une fonction reduire(a) qui utilise la dimension comme la variable et nous détectons le changement de taux d'erreur lorsque la dimension est réduite à entre 0 et 400. On peut constater que lorsque la dimension est égale à 0, le taux d'erreur est le plus élevé, puis diminue progressivement pour se stabiliser finalement environ 150 (valeur finalement environ 0.498).




(En fait ici on a une petite question : Si on ne fait pas la réduction de la dimension, donc la dimension reste encore 784, le taux d'erreur est 0.3242, mais une fois qu'on réduit la dimension, même de 784 à 780, le taux d'erreur augmente directement à environ 0.498, pour quoi ça ne présente pas comme un changement linéaire ?)

## Résultat obtenu :

On a trouvé que plus la dimension est retenue, plus le taux d'erreur est faible. Ce phénomène est en ligne avec nos règles cognitives.

Lorsque la dimension est réduite à 3,
Le temps de l'apprentissage est de 0.2451 sec.
Le temps de la classification est de 0.1670 sec.

La dimension est réduite à 30,
Le temps de l'apprentissage est de 0.3699 sec.
Le temps de la classification est de 0.2779 sec.

La dimension est réduite à 300,
Le temps de l'apprentissage est de 1.1200 sec.
Le temps de la classification est de 0.6881 sec.

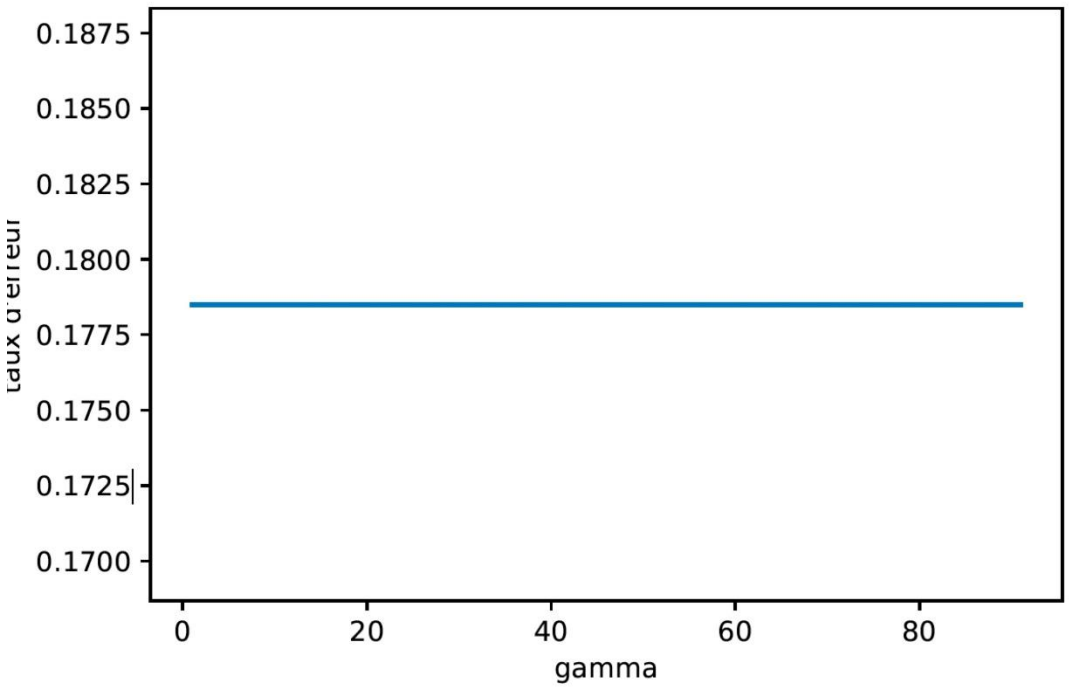
Fondamentalement, la tendance des dimensions les plus longues est plus longue. En général, plus la dimension est grande, plus le temps d'exécution est long.

## Partie3 : Support Vector Machines (SVM)

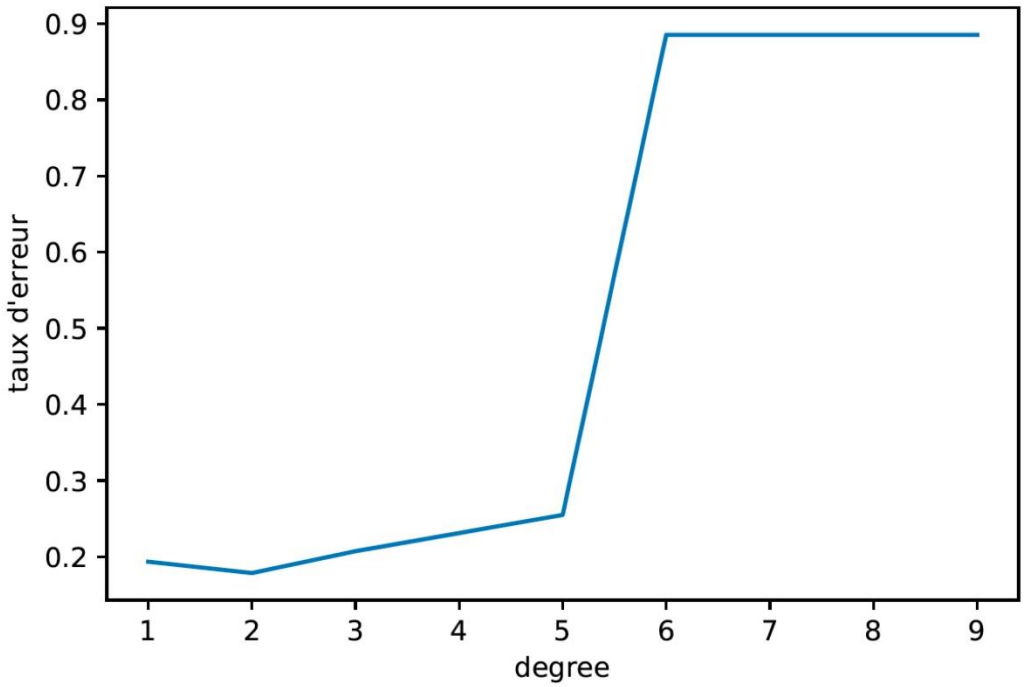
Question : Implémentation d'un ou plusieurs classifieurs de Scikit-Learn choisis en particulier parmi les Support Vector Machines (sklearn.svm) et les plus proches voisins (sklearn.neighbors). Vous tracerez la performance du classifieur en fonction de différentes configurations que vous choisirez ainsi que les matrices de confusion des meilleurs systèmes (avec ou sans ACP préalable).

### Implémentation :

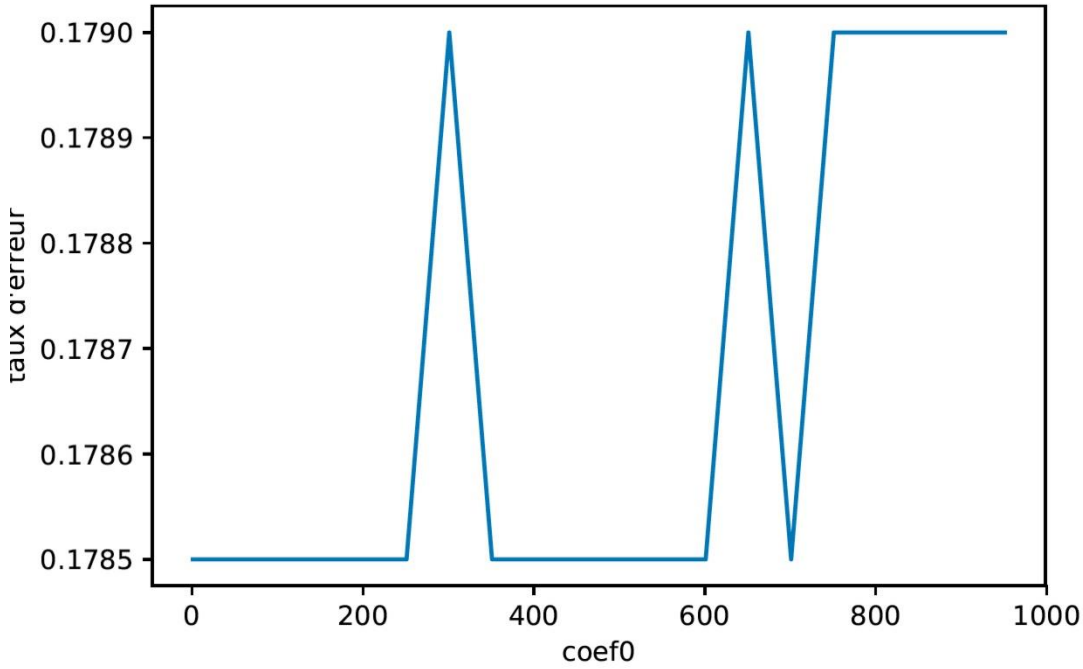
Nous avons constaté que lorsque kernel = 'rbf', le taux d'erreur reste inchangé à 0.9035, quelle que soit la manière dont les paramètres sont ajustés. Cette performance est très mauvaise, nous avons donc remplacé kernel = 'poly'.




Le paramètre gamma n'affecte pas les performances, nous le fixons à 1




Quand le degré est 2, le taux d'erreur est le plus bas : 0.1785




En fonction du taux d'erreur avec le changement de coef0, nous fixons Coef0=0.

## Résultat obtenu :

Le taux d'erreur est 0.1785.
Le temps de l'apprentissage est de 17.8032 sec.
Le temps de la classification est de 16.0658 sec.

### Matrice de confusion :

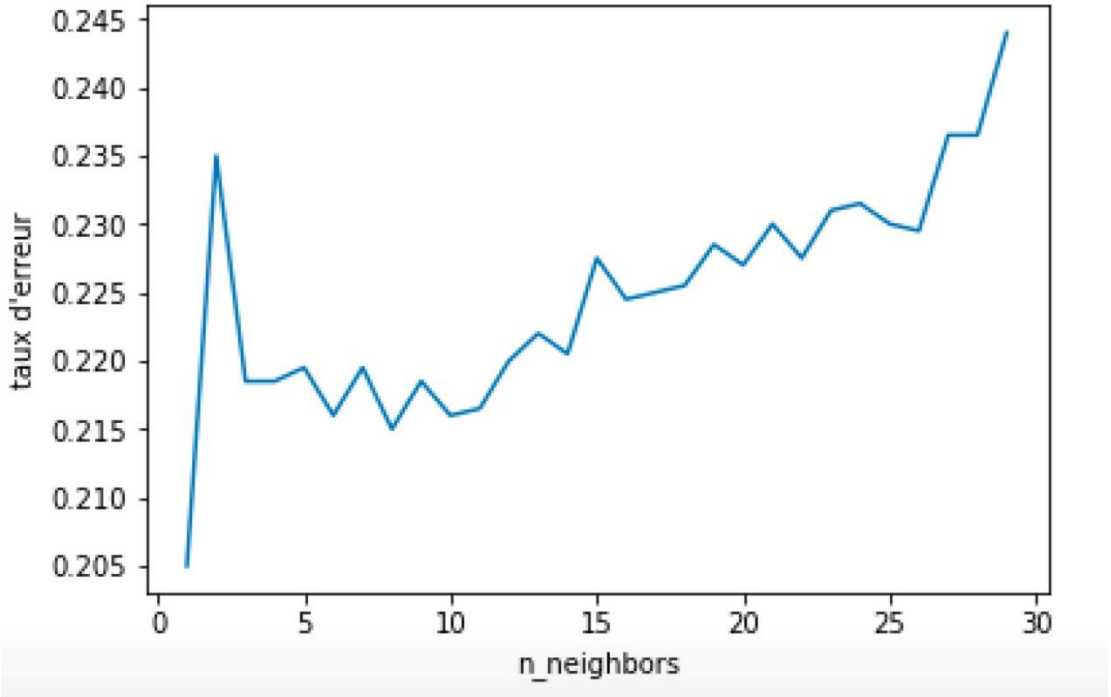
```
array([[389,   0,   4,  24,   1,   0,  80,   0,   5,   0],
       [  3, 497,   0,   9,   2,   0,   1,   0,   0,   0],
       [ 11,   0, 367,   3,  40,   0,  44,   0,   3,   0],
       [ 23,   7,  12, 410,  19,   0,  12,   0,   1,   0],
       [  2,   0,  63,  21, 380,   0,  41,   0,   2,   0],
       [  1,   0,   0,   1,   0, 466,   0,  23,   1,  11],
       [ 65,   0,  50,  17,  55,   0, 334,   0,   6,   0],
       [  0,   0,   0,   0,   0,   6,   0, 443,   1,  15],
       [  3,   0,   8,   0,   2,   5,   5,   1, 472,   0],
       [  1,   0,   0,   1,   0,   9,   0,  26,   1, 495]])
```

## Partie4 : Plus Proches Voisins (K Neighbors)

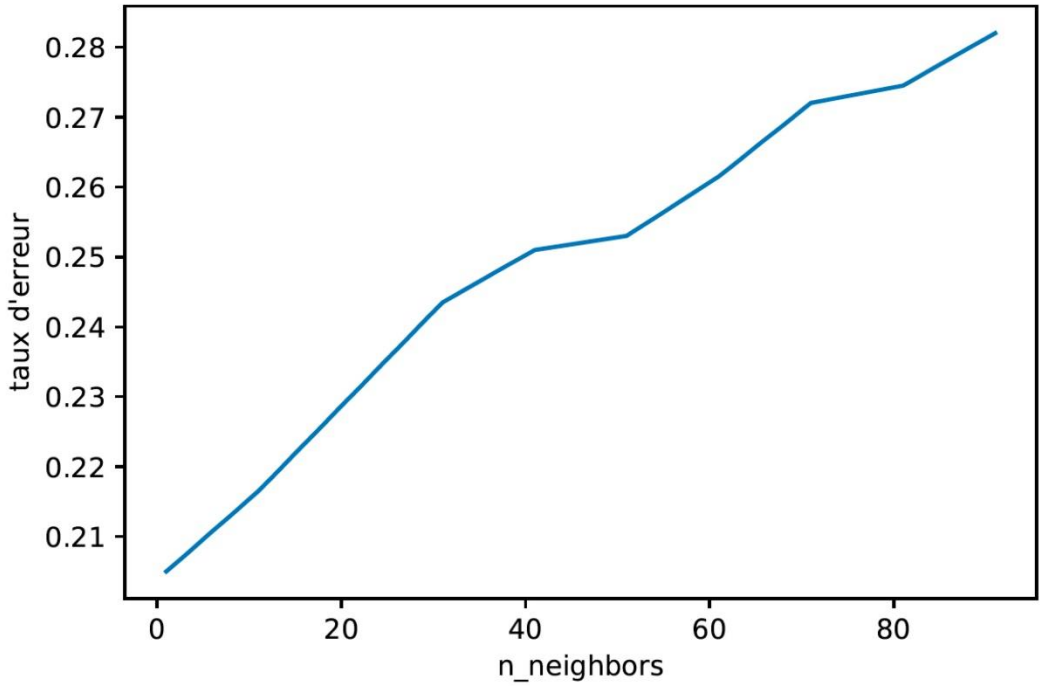
### Implémentation :

Cette méthode comparer les coordonnés d'une image avec ses n plus proches voisins pour prédire à quelle classe qu'il appartient. On a choisi de la paramétrer « nombre de voisin » pour tester ce classifieur, et de voir avec quel nombre de voisin on peut obtenir le meilleur résultat. Tout d'abord on a vu les résultats si le nombre de voisins n = 1. Ensuite on l'a augmenté pour essayer d'obtenir le n optimal.

(n entre 1 et 30, divisé par 1)



(n entre 1 et 100, divisé par 10)




On a constaté que une fois qu'on augmente le nombre de voisin, le temps d'exécution augmente un peu, et le taux d'erreur diminue d'abord jusqu'à n = 10, puis ça augmente encore une fois.

Ce résultat est assez logique parce qu'au début quand on augmente le nombre de voisins, on augmente au même temp les points de comparaison proches, donc il a plus de possibilité de trouver la classe exacte. Mais si on prend trop de voisins, on compare l'élément à classifier avec des éléments qui sont très loin de ceci, qui ne sont donc plus cohérents pour la classification, donc le taux d'erreur a vachement augmenté.

## Meilleur système :

En fonction des résultats obtenus, notre choix du système est la méthode SVM sans PCA. Parce que l'on peut avoir le taux d'erreur le plus faible, et le temps d'exécution n'est pas excessif.

## Conclusion du projet :

Ce projet nous permet de mieux comprendre le Machine Learning et les méthodes souvent utilisées. Nous avons appliqué la théorie que nous avons appris à pratiquer. On a bien étudié les principes des méthodes de classification et leurs différences.

En plus, ce projet nous permet de développer nos connaissances en Python. Nous pouvons maintenant mieux utiliser Python pour analyser les données.