Informatique PT. 2026 - 2027

Mathématiques & Informatique - F. Gaunard
`http://frederic.gaunard.com`
Lycée Voltaire, Paris 11e.

4 Algorithmes d'apprentissage

L'*intelligence artificielle* est définie comme étant l'ensemble des théories et des techniques mises en œuvre afin de réaliser des machines capables de simuler l'intelligence du vivant. La faculté d'apprentissage étant l'une des caractéristiques de l'intelligence, l'intelligence artificielle recouvre en particulier les techniques permettant à des machines de réaliser des apprentissages dans le but d'améliorer leurs performances avec le temps : on parle d'*apprentissage automatique* ou d'*apprentissage statistique* (*machine learning* en anglais).

On distingue deux type d'apprentissage : le plus fréquent, l'apprentissage supervisé, consiste pour un opérateur à montrer à la machine des milliers voire des millions d'exemples étiquetés avec leur catégorie, qui permettront à la machine de déterminer elle-même les paramètres pertinents pour classer chaque objet dans la catégorie qui lui correspond. Une fois cette phase d'apprentissage terminée, la machine doit être capable de généraliser à des objets non encore vus.

L'apprentissage non supervisé est plus ambitieux, mais il est aussi plus proche de notre propre modèle d'apprentissage, basé sur l'observation. Il consiste à faire en sorte qu'à partir d'un ensemble de données la machine soit capable de créer ses propres catégories, et si possible que ces catégories soient pertinentes pour nous...

Comme exemples d'appplication de l'apprentissage automatique, on peut citer : la reconnaissance de caractères manuscrits, la reconnaissance d'un individu sur une photo, la conduite d'un véhicule autonome, la traduction automatique d'un texte dans une autre langue ...

Dans ce chapitre nous allons nous intéresser à deux algorithmes utilisés dans ces deux modes d'apprentissage : l'algorithme des k plus proches voisins (ou KNN), et l'algorithme des k moyennes (ou KM).
Ces deux algorithmes permettent de répartir des données entre plusieurs catégories, en ce sens ce sont des algorithmes de classification.

1 Une méthode d'apprentissage supervisé : l'algorithme des k plus proches voisins

1.1 Partitionnement de données

Nous allons considérer un ensemble de points de coordonnées $(x, y) \in [-1,5; 1,5]^2$, qui peuvent appartenir à deux catégories représentées par des triangles bleus et des carrés rouges et nous allons fournir à la machine un jeu de 250 données d'entrainement (voir la *Figure 1*).

Les carrés rouges correspondent aux points situés à l'intérieur du cercle trigonométrique, et les triangles bleus aux points situés à l'extérieur du cercle trigonométrique. Mais ni la machine ni le programmeur ne le savent.

Une fois ces données acquises nous souhaitons qu'à partir d'un point de coordonnées $(x, y) \in [-1,5; 1,5]^2$, la machine lui attribue une catégorie (bleu ou rouge).

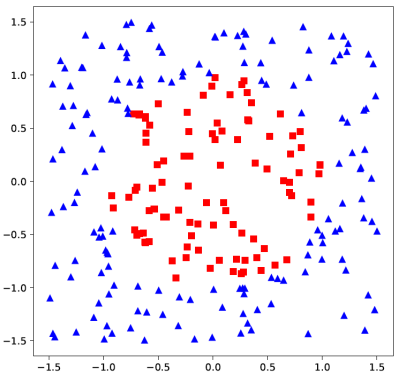


FIGURE 1.

L'algorithme des k plus proches voisins

La méthode des k plus proches voisins consiste à déterminer les k données d'entrainement les plus proches du point (x, y), et à attribuer à ce point la catégorie majoritaire parmi ces voisins.

Dans l'algorithme qui suit, nous allons supposer que `DonneesApprentissage` est un tableau contenant les données d'apprentissage sous la forme (x, y, c) avec $(x, y) \in [-1,5; 1,5]^2$ et $c =$`"blue"` ou $c =$`"red"`.

Pour ce faire, on commence par définir une fonction Distance qui renvoie **la fonctio**n qui calcule le carrée de la distance d'un point à M.

```
def Distance(M):
    def d(P):
        return (P[0]-M[0])**2+(P[1]-M[1])**2
    return d
```

L'algorithme des k plus proches voisins (KNN), prend en paramètre un point M et un entier k puis fonctionne comme suit

- ✗ on commence par trier le tableau `DonneesApprentissage` avec la commande `sorted` (qui peut le faire en fonction de n'importe quelle clé, c'est-à-dire de n'importe quelle grandeur associée à chacun de ses termes : on utilise ici la distance au point M);
- ✗ on on compte, parmi les k plus proches, combien sont rouges.
- ✗ si plus de la moitiés des k plus proches voisins sont rouges, on renvoie `'red'`, sinon `'blue'`.

```
def PlusProchesVoisins(M,k):
    liste=sorted(DonneesApprentissage, key=Distance(M))
    r=0
    for d in liste[:k]:
        if d[2]=='red' :
            r=r+1
    if r>k//2:
        return 'red'
    else:
        return 'blue'
```

On peut alors utiliser cette fonction pour colorier en rouge ou en bleu un **nouveau** jeu de données, sous la forme d'une liste `DonneesTest` de longueur 1000 contenant des couples aléatoires $(x, y) \in [-1,5; 1,5]^2$.
Sur la *Figure 2* sont représentées les catégories attribuées à chacun des points par cet algorithme pour $k = 3$ et pour $k = 7$. On a également représenté le cercle trigonométrique, afin de pouvoir évaluer à l'oeil la correction de l'algorithme.

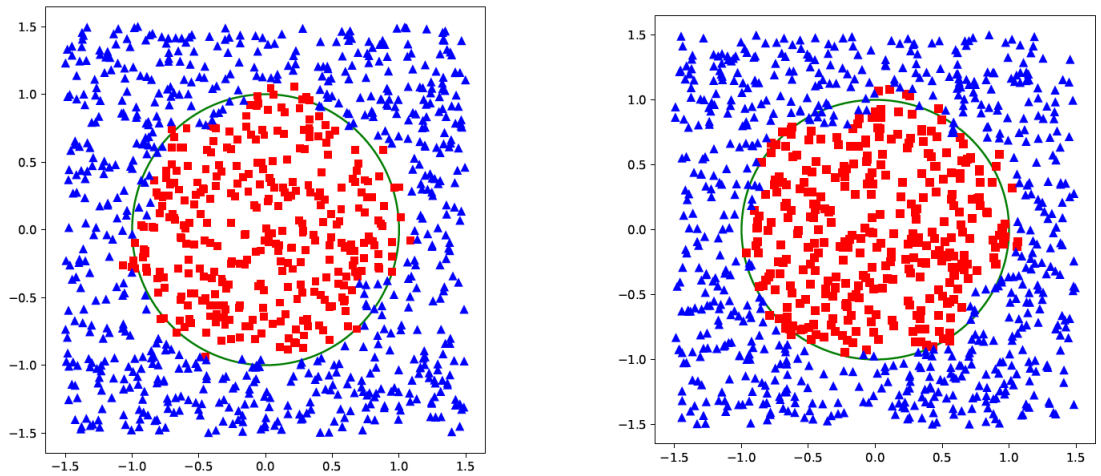


FIGURE 2.

On joint le code utilisé pour tracer les graphes ci-dessus. On utilise la fonction `uniform` de la bibliothèque `random` via le code `random.uniform(a,b)` qui renvoie un flottant choisi aléatoirement dans l'intervalle $[a, b]$.

```
import random as rd

DonneesTest=[]
for k in range(1000):
    x,y=rd.uniform(-1.5,1.5),rd.uniform(-1.5,1.5)
    DonneesTest.append((x,y))

listexr=[]
listexb=[]
listeyr=[]
listeyb=[]

for M in DonneesTest:
    if PlusProchesVoisins(M,7)=='red':
        listexr.append(M[0])
        listeyr.append(M[1])
    else:
        listexb.append(M[0])
        listeyb.append(M[1])

listet=[2*ma.pi*k/1000 for k in range(1001)]
plt.plot([ma.cos(t) for t in listet],[ma.sin(t) for t in listet],color='green')
plt.plot(listexr,listeyr,'s',color='red')
plt.plot(listexb,listeyb,'^',color='blue')
```

Matrice de confusion

Comment interpréter les résultats que nous avons obtenus? Dans un problème d'apprentissage automatique, on sépare les données en deux sous-ensembles : les données réservées à l'apprentissage et les données destinées à tester la qualité de l'apprentissage.

Nous allons donc supposer que l'on dispose d'un tableau test contenant 100 autres données, toujours sous la forme (x, y, c) avec $(x, y) \in [0, 1]^2$ et $c =$`"blue"` ou $c =$`"red"` et nous allons appliquer la méthode du plus proche voisin à chacun de ces points. Si notre objectif est de reconnaitre si un point est rouge ou pas, les points tests vont se répartir en quatre catégories :

- ✗ les vrais positifs VP (les points rouges reconnus comme tels);
- ✗ les vrais négatifs VN (les points bleus reconnus comme tels);
- ✗ les faux positifs FP (les points bleus reconnus à tort comme rouges);
- ✗ les faux négatifs FN (les points rouges reconnus à tort comme bleus).

Ces quatre valeurs sont rangées dans une matrice de la forme $\mathrm{C} = \left(\begin{array}{c|c} \mathrm{VP} & \mathrm{FN} \\ \hline \mathrm{FP} & \mathrm{VN} \end{array}\right)$ appelée matrice de confusion.

Voici par exemple les matrices que l'on obtient avec différentes valeurs de k, pour le même jeu de données et de test :

k	1	3	5	7	11
C	$\begin{pmatrix} 32 & 2 \\ 1 & 65 \end{pmatrix}$	$\begin{pmatrix} 32 & 2 \\ 0 & 66 \end{pmatrix}$	$\begin{pmatrix} 32 & 2 \\ 2 & 64 \end{pmatrix}$	$\begin{pmatrix} 30 & 4 \\ 2 & 64 \end{pmatrix}$	$\begin{pmatrix} 31 & 3 \\ 3 & 63 \end{pmatrix}$

La matrice de confusion peut aider à choisir la meilleure valeur de k : sur cet exemple on choisira plutôt $k = 3$, mais tout dépend de l'exemple.

Reconnaissance de caractères

Dans la pratique, on n'utilise pas la totalité des échantillons pour l'apprentissage : une partie est préservée pour servir à *tester* la qualité de la reconnaissance.
Dans l'exemple qui suit, nous disposons de 1797 images de 8×8 pixels en niveau de gris représentant des chiffres de 0 à 9 (un extrait de la base de données est présenté *Figure 3*).
Nous allons utiliser la méthode des $k-$voisins (étendue à plus de deux catégories) pour obtenir un *classifieur* permettant de déterminer à quel chiffre correspond une nouvelle image.
Une image *digitalisée* est donc un vecteur $I \in \mathbb{R}^{64}$. La distance entre deux images I et J est définie comme

$$\sum (I_i - J_i)^2$$

où i parcourt les 64 pixels.

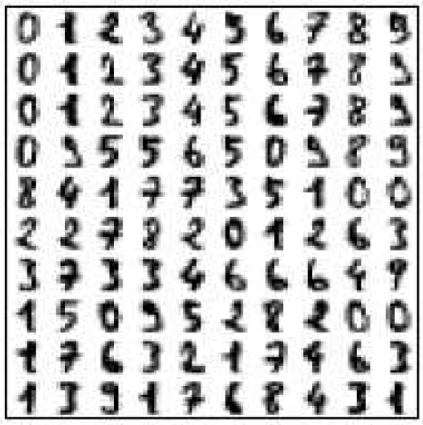


FIGURE 3.

Plus précisément, ces données nous sont fournies par l'intermédiaire de deux tableaux :

- ✗ X est une matrice de taille 1797×64; pour tout $k \in [\![0, 1796]\!]$, $X[k]$ est un vecteur de $\mathbb{R}^{64}$ qui représente l'image digitalisée d'un chiffre;
- ✗ Y est un vecteur de taille 1797 ; pour tout $k \in [\![0, 1796]\!]$, $Y[k]$ est le label de $X[k]$, autrement dit l'entier compris entre 0 et 9 qui est représenté par $X[k]$.

Nous allons partager ces données en deux parties sensiblement égales ; la première consacrée à l'apprentissage, la seconde au test.
Dans la partie consacrée à l'apprentissage, on va choisir 90 représentants de chacune de nos dix classes[1], il restera donc 897 données pour nous servir de test.

Une fois l'algorithme des k plus proches voisins implémenté comme ci-avant, trois cas de figure sont possibles pour chacune des données de l'échantillon test :

- ✗ la fonction `PlusProchesVoisins` renvoie une seule valeur, qui est exacte;
- ✗ la fonction `PlusProchesVoisins` renvoie une seule valeur, qui est inexacte (la machine se trompe);
- ✗ la fonction `PlusProchesvoisins` renvoie plusieurs valeurs (la machine est indécise, car parmi les k plus proches voisins il y a deux chiffres qui sont représentés le même nombre maximal de fois).

Pour $k = 3$, sur les 897 valeurs de test on constate que:

- ✗ la machine reconnait le bon caractère 858 fois (soit un taux de réussite de plus de 95%);
- ✗ la machine se trompe 31 fois;
- ✗ la machine est indécise 8 fois.

Les erreurs que la machine a commise sont présentées *Figure* 4 ; difficile de lui reprocher quoi que ce soit :

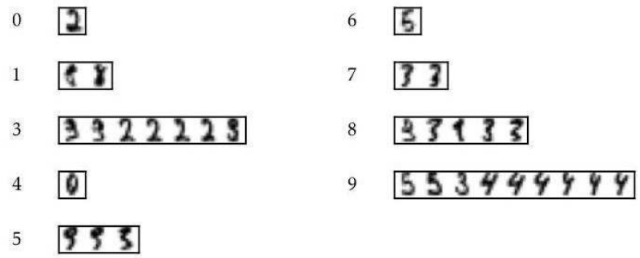


FIGURE 4.

[1]Les données d'apprentissage doivent être choisies de façon *équirépartie*, et ne pas être trop nombreuses pour éviter le phénomène de *surapprentissage*

Améliorations possibles

La méthode que nous venons de présenter est bien entendu simpliste, mais peut être améliorée de plusieurs façons, par exemple :

- ✗ au lieu de considérer les k plus proches voisins, considérer tous les voisins contenus dans une boule de rayon ε centrée autour de l'objet à classer;
- ✗ pondérer la contribution de chaque voisin par un coefficient inversement proportionnel à la distance à l'objet à classer;
- ✗ utiliser d'autres distances que la distance euclidienne (distance de *Manhattan*, distance de *Tchebychev*...)

Exercices

Exercice 1.

Soit les points de coordonnées suivantes:

$$A(1,6), B(2,6), C(3,1), D(4,2), E(6,0), F(7,5), G(7,3), H(10,3).$$

1. En utilisant la distance euclidienne, quels sont les deux plus proches voisins du point $P(5,5)$?

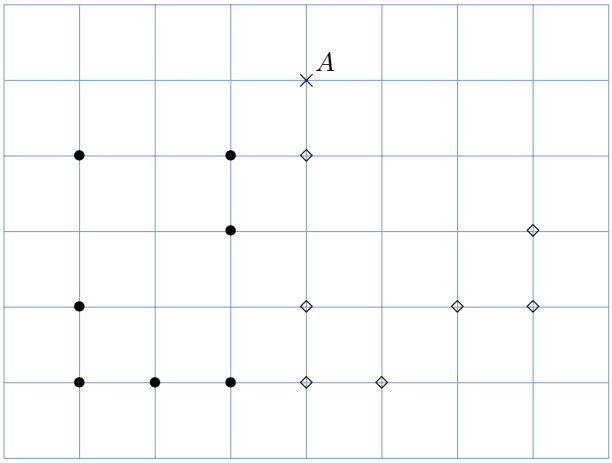
On définit la *distance de Manhattan* entre deux points $M_1(x_1, y_1)$ et $M_2(x_2, y_2)$ par

$$d_{\mathcal{M}}(M_1, M2) = |x_1 - x_2| + |y_1 - y_2|.$$

2. Quels sont les deux plus proches voisins de P pour $d_{\mathcal{M}}$?

Exercice 2.

Dans le quadrillage ci-dessous 14 points sont dessinés, dont 7 de la classe C_1, avec des ronds noirs $\cdot$, et 7 de la classe C_2, avec des losanges $\diamond$.
On introduit un nouveau point A , dont on cherche la classe à l'aide d'un algorithme des k plus proches voisins pour la distance euclidienne, en faisant varier la valeur de k parmi 1,3 et 5.
Quelle est la bonne réponse (sous la forme d'un triplet de classes pour le triplet $(1,3,5)$ des valeurs de k)?



Exercice 3. Les fruits et les couleurs

On cherche à prédire la couleur d'un fruit en fonction de sa largeur (L) et de sa hauteur (H). On dispose des données d'apprentissage suivantes :

largeur	hauteur	couleur
2	6	red
5	6	yellow
2	5	orange
6	5	purple
1	2	red
4	2	blue
2	1	violet
6	1	green

Soit U le nouveau fruit de largeur $L = 1$, et de hauteur $H = 4$.

1. Placer ces données dans un repère (L en abscisse, H en ordonnée).
   Indiquez pour chaque point sa couleur.
2. Quelle est la de U couleur si l'on considère 1 voisin? si l'on considère 3 voisins?
3. Plutôt que le vote majoritaire, on voudrait considérer le vote des voisins pondérés par la distance. Chaque voisin vote selon un poids ω inversement proportionnel au carré de sa distance : $\omega = 1/d^2$. On prend 3 voisins, quelle est la couleur de U ? Comparez vos résultats à ceux de la Question **2.**

2 Une méthode d'apprentissage non supervisé : l'algorithme des k moyennes

2.1 Partitionnement de données

Nous allons maintenant nous intéresser à un problème plus complexe : comment comprendre la structure des données **sans aider** la machine en lui donnant des données d'apprentissage étiquetées?

En pratique il s'agit pour la machine de regrouper les données proches au sein d'une même classe (ou *cluster* en anglais), à charge pour l'humain d'interpréter ensuite ce regroupement.

Par exemple, en regardant la *Figure 5*, nous voyons tous trois classes de points. Nous devons faire en sorte qu'il en soit de même de la machine. Nous allons quand même l'aider un peu en lui imposant le nombre k de classes. En appelant $\mathrm{C}_1, \dots, \mathrm{C}_k$ ces classes on note

- ✗ $P_j = \dfrac{1}{\mathrm{card}\mathrm{C}_j} \displaystyle\sum_{x \in \mathrm{C}_j} x$ le *barycentre* de la classe C_j ;
- ✗ $m_j = \displaystyle\sum_{x \in \mathrm{C}_j} \|x - P_j\|^2$ le *moment d'inertie* de la classe C_j.

L'objectif de la machine va être de regrouper les données en k classes $C_1, \dots, C_k$ de sorte que la somme $\displaystyle\sum_{j=1}^{k} m_j$ des moments d'inertie soit minimale.

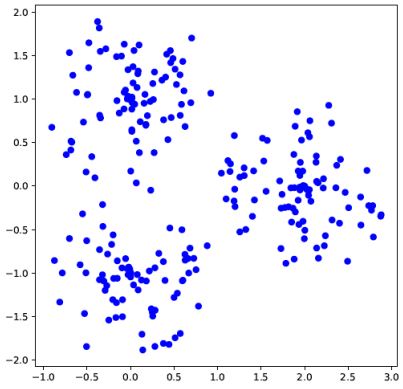


FIGURE 5.

L'algorithme des k moyennes

Le calcul de la classification optimale, c'est-à-dire minimisant la somme des moments d'inertie, est un problème très coûteux en temps, aussi allons nous employer un algorithme *glouton*.
Ce dernier nous assurera d'obtenir in fine une « bonne » solution, mais pas forcément la meilleure. L'algorithme des k moyennes consiste à réaliser la succession d'opérations suivantes, tant que les points $P_1, \dots, P_k$ évoluent :

i. on choisit *aléatoirement* k centres $P_1, \dots, P_k$ (parfois appelés *centroïdes*);
ii. chacun des points du nuage est associé au centre P_j le plus proche, ce qui crée k classes $\mathrm{C}_1, \dots, \mathrm{C}_k$;
iii. on calcule les barycentres $P_1, \dots, P_k$ de ces classes, qui remplacent les valeurs précédentes;
iv. on reprend le calcul à partir de l'étape *ii.*

Nous admettrons que cet algorithme converge, autrement dit qu'à partir d'une certaine étape les barycentres $P_1, \dots, P_k$ ne se déplacent plus et donc que les classes $C_1, \dots, C_k$ sont stabilisées.

Cet algorithme a l'avantage d'être simple à implémenter, et de complexité raisonnable, mais a pour inconvénient de donner une configuration pour laquelle la somme des moments d'inertie n'est qu'un minimum local, mais pas forcément un minimum global : dans certains cas, la somme des moment d'inertie n'est pas minimale.

La *Figure 6* représente le résultat de cet algorithme pour $k = 3$ appliqué à l'ensemble de points précédents.

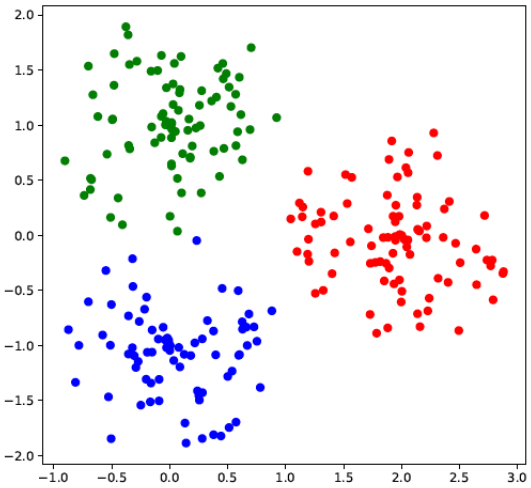


FIGURE 6.

On joint le code implémentant l'algorithme des k moyennes qui permet d'obtenir la figure ci-dessus.
On utilise la fonction `randint` de la bibliothèque `random` : `random.randint(n,p)` renvoie un entier choisi suivant la loi uniforme dans l'intervalle $[\![n, p]\!]$.

La fonction ci-dessous calcule le barycentre d'une liste de points.

```
def Barycentre(C):
    # Calcule le barycentre d'une liste de points
    S=[0,0]
    for N in C :
        S=[S[0]+N[0],S[1]+N[1]]
    return [S[0]/len(C),S[1]/len(C)]
```

Celle qui suit détermine l'indice du point de la liste `Barycentres` prise en arguments qui est le plus proche de M également pris en argument, en comparant les distances de M à chacun des points de la liste.

```
def PlusProche(M, Barycentres):
    n=len(Barycentres)
    P=Barycentres[0]
    IndiceCandidat=0
    for k in range(1,n):
        P2=Barycentres[k]
        if (M[0]-P2[0])**2+(M[1]-P2[1])**2<(M[0]-P[0])**2+(M[1]-P[1])**2:
            P=P2
            IndiceCandidat=k
    return IndiceCandidat
```

Puis, l'algorithme des $k-$moyennes renvoie la liste des *barycentres stabilisés*.
Les premiers barycentres sont choisis aléatoirement dans le nuage de points.
En supposant que $k << n$, la probabilité que `rd.randint(0,n)` est très très faible (on peut considérer qu'elle est nulle).
La classe d'un point est l'indice du barycentre le plus proche de la liste des barycentres.

```
def KMoyennes(Donnees, k):
    n=len(Donnees)
    Barycentres=[Donnees[rd.randint(0,n-1)] for i in range(k)]
    Different=True
    while Different : # tant que ce n'est pas stable
        C = [ [ ] for i in range(k)] # on crée des classes vides
        for i in range(n): # on parcourt tous les points
            j=PlusProche(Donnees[i],Barycentres) # on récupère la classe du point
            C[j]=C[j]+[Donnees[i]] # on rajoute le point dans la classe
        Barycentres_new = [Barycentre(C[i]) for i in range(k)]
        # On reprend le barycentre de chaque nouvelle classe
        if Barycentres==Barycentres_new:
            Different=False
        Barycentres=Barycentres_new
    return Barycentres
```

Enfin, il ne reste plus qu'à former les classes autour de chacun des barycentres stabilisés et à les "colorier".

```
n=len(Donnees)
Barycentres=KMoyennes(Donnees,3)
ListeR=[ ]
ListeB=[ ]
ListeV=[ ]
for i in range(n):
    j=PlusProche(Donnees[i],Barycentres)
    if j==0:
        ListeR.append(Donnees[i])
    elif j==1:
        ListeB.append(Donnees[i])
    elif j==2:
        ListeV.append(Donnees[i])

plt.plot([P[0] for P in ListeR],[P[1] for P in ListeR],'o',color='red')
plt.plot([P[0] for P in ListeB],[P[1] for P in ListeB],'o',color='blue')
plt.plot([P[0] for P in ListeV],[P[1] for P in ListeV],'o',color='green')
plt.show()
```

2.2 Reconnaissance de caractères

Considérons de nouveau nos 1797 images de chiffres de 0 à 9, mais cette fois-ci sans prendre en compte leur label. Nous allons appliquer l'algorithme des k moyennes avec $k = 10$, de manière à obtenir un regroupement de ces images en 10 classes, qu'on espère correspondre aux différents chiffres représentés.
Voici les images (*Figure 7*) correspondants aux barycentres des dix classes que l'on obtient par cet algorithme pour $k = 10$:

FIGURE 7.

Manifestement, les classes obtenues correspondent bien à notre souhait. Si on passe maintenant en revue toutes les images de chacune de ces dix classes en tenant compte de leur label, on obtient 1426 succès et 371 échecs. Autrement dit, 79% des images ont étés rangées dans les bonnes classes. Plus précisément, on peut pour chacune des 10 classes calculer le pourcentage de réussite :

Classe	0	1	5	7	8	4	3	2	9	6
Réussite	99%	60%	91%	85%	45%	98%	87%	85%	56%	97%

Le plus mauvais score est obtenu pour la classe de 8 qui ne comporte que 45% d'images représentant le chiffre 8 ; à l'inverse, la première classe comporte 99 % d'images représentant le chiffre 0.