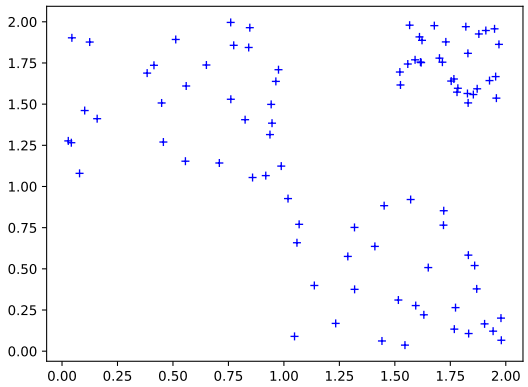
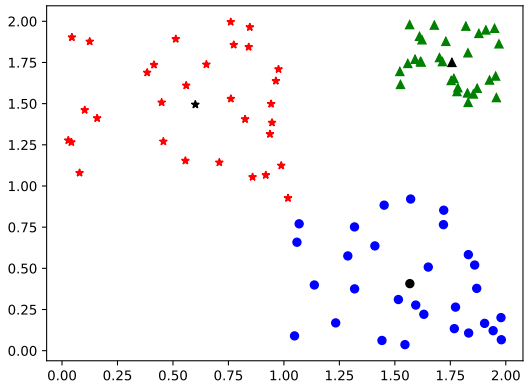
Algorithme des k–moyennes

Contexte du chapitre

On dispose d'un certain nombre de points, issus de plusieurs catégories. **On ignore quel point appartient à quelle catégorie**. Comment retrouver les catégories ?





I Géométrie

Définition 1

Soit $x_1, \cdots, x_n$ des points d'un espace préhilbertien E, on appelle **isobarycentre** le point $\mu := \frac{1}{n}\sum_{i=1}^{n} x_i$. On parlera dans ce cours de **barycentre** pour désigner l'isobarycentre.

La valeur $\sum_{i=1}^{n} ||x_i - \mu||^2$ est alors **le moment d'inertie** du système de points $\{x_1, \cdots, x_n\}$.

Remarque On a ici une généralisation de la moyenne empirique et de la variance empirique de l'échantillon $(x_1, \cdots, x_n)$.

p.8 **Exercice 1** On suppose $x_1, \cdots, x_n$ des points d'un espace vectoriel E préhilbertien. Montrer que le barycentre μ est le point unique où la fonction $f : \begin{array}{ccc} E & \longrightarrow & \mathbb{R}_+ \\ m & \longmapsto & \sum_{i=1}^{n} ||x_i - m||^2 \end{array}$ atteint son minimum absolu.

Modélisation Informatique On représente un point de $\mathbb{R}^p$ par un tableau numpy de taille $(1, p)$. Désormais on comment l'identification « point = tableau numpy ».
Par exemple, si on parle de liste de points, on parle en fait d'une liste Python dont les éléments sont des tableaux numpy (1,p).

p.8 **Exercice 2 En utilisant sans limite toutes les possibilités de la bibliothèque numpy :**

1. Ecrire une fonction distance_carre qui, lorsque deux points A et B sont passés en argument, retourne $||A-B||^2$.
2. Ecrire une fonction `barycentre` qui prend en argument une liste de points `listeX` et retourne le barycentre de la liste.
3. Ecrire une fonction `moment` qui prend en argument une liste de points `listeX` et retourne son moment d'inertie.
4. Ecrire une fonction `indice_plus_proche` qui reçoit en argument un point X et une liste de points liste ; et qui renvoie l'indice du points de la liste le plus proche de X. Par exemple :
   `indice_plus_proche(np.array([1,2,1]),[np.array([-5,2,2]),np.array([1,0,2]),np.array([1,2,4])])` retourne l'indice 1 car les distances respectives (au carré) sont $37, 5, 9$.
5. Comparer les deux commandes :

```
[1,2,3]==[1,2,3]

np.array([1,2,3])==np.array([1,2,3])
```

   Ecrire une fonction qui teste si deux points passés en argument sont bien égaux.

p.8 **Exercice 3** Que va produire le code suivant ?

```
def dessine_donnees(listeX, montrer=True):
    for A in listeX:
        x,y=A
        plot(x,y, 'b+' )
    if montrer:
        plt.show()

listeXX=[np.random.random(2)+[0,1] for _ in range(30)]+[np.random.random(2)+[1,0] for _ in range(30)]
+[0.5*np.random.random(2)+[1.5,1.5] for _ in range(30)]

dessine_donnees(listeXX)
```

II Algorithme des k–means (ou des k–moyennes)

1 Formalisation du problème

On dispose de $X_1, \ldots, X_n$ des points de $\mathbb{R}^p$, on souhaite des répartir en k classes $C_1, C_2, \ldots, C_k$ qio forment une **partition** de l'ensemble $\{x_1, \ldots, x_n\}$. On aura ainsi : $C_1 \cup C_2 \cup \ldots \cup C_k = \{X_1, \ldots, X_n\}$ et $\forall i, i' \ (i \neq i' \Rightarrow C_i \cap C_{i'} = \varnothing)$. On note c_j le centre de la classe C_j.

2 Algorithme

1. (a) Prendre k centres $c_1, \cdots, c_k$ au hasard parmi les points.
   (b) Initialiser les classes $C_1, C_2, \cdots, C_k := \varnothing$
   (c) Pour chaque point X : trouver j tel que $d(X, c_j)$ est minimum ; faire $C_j \to C_j \cup \{X\}$
   (d) Pour chaque j : calculer $c'_j = \dfrac{1}{|C_j|} \displaystyle\sum_{X \in C_j} X$ le barycentre de la classe C_j.
2. Tant que $(c_1, \ldots, c_k) \neq (c'_1, \ldots, c'_k)$:
   (a) $(c_1, \ldots, c_k) := (c'_1, \ldots, c'_k)$
   (b) Initialiser les classes $C_1, C_2, \cdots, C_k := \varnothing$
   (c) Pour chaque point X : trouver j tel que $d(X, c_j)$ est minimum ; faire $C_j \to C_j \cup \{X\}$
   (d) Pour chaque j : calculer $c'_j = \dfrac{1}{|C_j|} \displaystyle\sum_{X \in C_j} X$ le barycentre de la classe C_j.

3 Programme

```
# teste si deux listes de points sont égales
def listes_egales(l1,l2):
    if not len(l1)==len(l2):
        return False
    for i in range(len(l1)):
        if not(l1[i]==l2[i]).all():
            return False
    return True

# pour l'initialisation
# tirer une p-liste dans une liste au hasard (avec utilisation d'un dictionnaire)

def p_liste_alea(liste,p):
    d={}
    N=len(liste)
    while len(d)<p:
        cle=np.random.randint(N)
        if cle not in d:
            d[cle]=True
    return [liste[cle] for cle in d]



#une itération des k means:
def mise_a_jour_centres(centres,listeX):
    k=len(centres)
    nouveaux_centres=[np.zeros(2) for _ in range(k)]
    cardinal_classes=[0 for _ in range(k)]
    for X in listeX:
        nouvelle_classe_de_X=indice_plus_proche(X,centres)
        nouveaux_centres[nouvelle_classe_de_X]+=X
        cardinal_classes[nouvelle_classe_de_X]+=1
    nouveaux_centres=[nouveaux_centres[i]/cardinal_classes[i] for i in range(k)]
    return nouveaux_centres


#k means:
def kmoyennes(listeX,k):
    centres=p_liste_alea(listeX,k)
    nouveaux_centres=mise_a_jour_centres(centres,listeX)
    while not (listes_egales(centres, nouveaux_centres)):
        centres=nouveaux_centres
        nouveaux_centres=mise_a_jour_centres(centres,listeX)
    return centres

#graphisme
def dessine_donnees_avec_moyennnes(listeX, centres):
    formes=['r*', 'bo', 'g^', 'ro', 'b^', 'g*']
    for k in range(len(centres)):
        allure=formes[k]
        plot(centres[k][0],centres[k][1],allure[1]+'k')
    for X in listeX:
```

```
        classe_de_X=indice_plus_proche(X,centres)
        plot(X[0],X[1], formes[classe_de_X])
    plt.show()
```

Ici si on éxécute :

```
listeXX=[np.random.random(2)+[0,1] for _ in range(30)]+[np.random.random(2)+[1,0] for _ in range(30)]

ccentres=kmoyennes(listeXX,3)

print(ccentres)
```

On trouve par exemple :

```
[array([1.72267616, 1.77789824]), array([0.57046023, 1.47787177]), array([1.53774592, 0.4362601 ])]
```

Ce qui n'est pas loin des trois centres théoriques qu'on a prévus initialement pour la simulation.

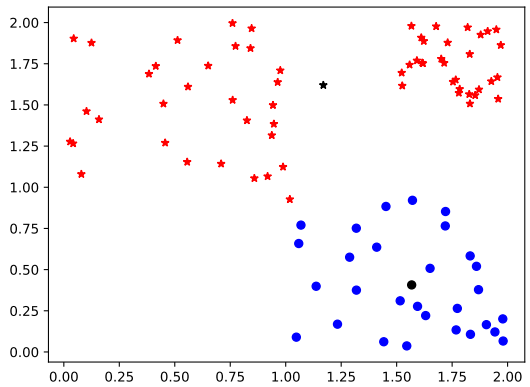
p.8 **Exercice 4** Ma fonction `liste_egales` est juste (elle teste bien l'égalité de deux listes de points), et pourtant il y a un problème.

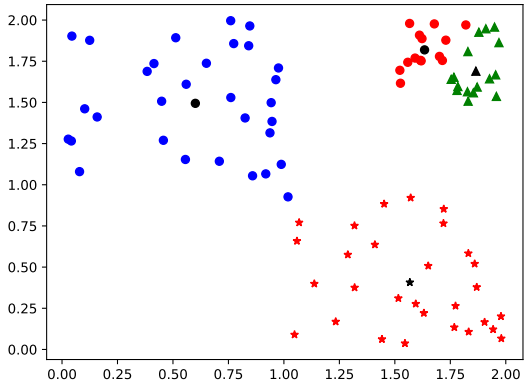
1. Voyez–vous lequel ?
2. Quelles solutions pour y remédier ?

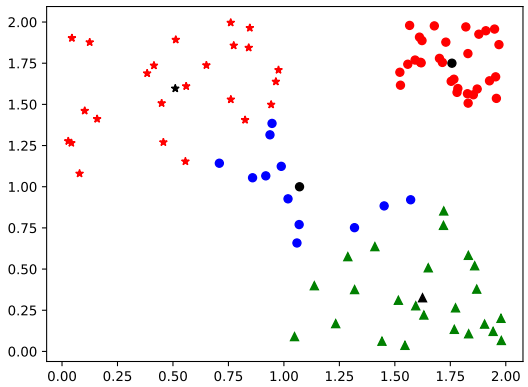
III Choix du nombre de classes

On savait a priori qu'il y avait trois classes, parce que l'on sait très bien comment on a généré les données. En pratique la question de la connaissance de k est en fait cruciale ; (c'est un peu à rapprocher de la détermination de k dans la méthode des k plus proches voisins.)

1 Exemples de résultats pour certains choix de k







Ici, on **sait** que la bonne réponse est $k = 3$.

1. Pour $k = 2$ le modèle proposé est trop simple et perd une partie de l'information.
2. Pour $k > 3$ les groupes créés sont, pour certains, artificiels : l'algorithme lit de l'information là où il n'y a que du bruit, comme un enfant lisant des visages écrits dans des nuages (ou toute autre superstition fausse, dont les adultes ne sont d'ailleurs pas exempts).

Un choix inadéquat de k, correspond à un sous/sur apprentissage. Mais comment savoir quel k choisir, si on ignore k à l'avance ?

2 Classification supervisée, classification non supervisée, risque empirique

a) Définitions

Revenons sur la méthode des k plus proches voisins ; gardons en tête la méthode des k–moyennes.
On dispose dans les deux cas de la notion de **risque empirique**. Ce risque est l'écart entre ce qui est prédit et ce qui est observé **pour les données servant à construire le modèle**.

$$\text{Risque empirique} = \frac{1}{n}\sum_{i=1}^{n} d(\text{valeur prédite du } i\text{éme point}, \text{valeur réelle du } i\text{éme point})$$

1. (Méthode des k plus proches voisins, rappel) Le risque empirique est le taux d'erreur de classification des données :

   $$R_e(k) = \text{ Risque empirique} = \frac{\text{Nombre de points mal classés}}{\text{Nombre total de points}}$$

   Lorsque k diminue, le risque diminue ; il est nul pour $k = 1$.

   Il ne faut pas le confondre avec le risque **théorique** :

   $$R_t(k) = \frac{\text{Nombre de points mal classés}}{\text{Nombre total de points}} = \mathbb{P}(X\text{est mal classé})$$

   qu'on peut évaluer en prenant un ensemble de points **différents de ceux ayant servi à construire** et pour lesquels on connaît la classification correcte.

2. A k fixé faisons fonctionner l'algorithme des k–moyennes. On peut calculer, lorsque k classes sont définies, le risque empirique :

$$R(k) := \frac{1}{n}\sum_{i=1}^{n} ||X_i - c_{j_x}||^2 = \frac{1}{n}\sum_{j=1}^{k}\sum_{X\in C_j} ||X - c_j||^2$$

Ce risque est décroissant en fonction de k. Il s'interprête aussi comme la dispersion du nuage de points autour des centres et s'assimile à une variance. Lorsque k vaut n, il y a exactement un point par classe, chaque point de la classe est donc le barycentre de la classe, aussi le risque est-il alors nul.

3. Similarité des deux situations : les valeurs de k où le risque empirique est minimal, sont aussi les valeurs de k pour lesquelles le modèle est le plus instable. Si l'on modifie quelques données, les prédictions faites vont beaucoup changer. A l'autre extrémité, les valeurs de k pour lesquelles le risque empirique est le plus grand, sont celles pour lesquelles le modèle colle le moins aux données. Le modèle est alors très stable ; mais cela ne dit rien de son caractère prédictif qui est généralement pauvre. Par exemple :

   - Méthode des k plus proches voisins, $k =$ nombre de données : on a alors la couleur majoritaire uniformément.
   - Méthode des k plus proches voisins, $k = 1$: on a une seule classe, la prédiction faite est simplement le barycentre. C'est très robuste, mais très peu détaillé.

     Cela revient à résumer toute une série de notes par sa moyenne sans distinguer matière/nature de l'épreuve/-progrès ou non...

   La méthode constituant à minimiser le risque empirique mène à un phénomène de surapprentissage. Modèle retenu = non prédictif et sans robustesse, ayant appris le bruit des données sans chercher à discriminer.

4. Dissimilarité des deux situations. Les k plus proches voisins relèvent de la classification supervisée, et les k-moyennes de la classification non supervisée.

   - **En classification supervisée, on connaît la « bonne réponse » pour certains points**, et on cherche à prédire la réponse pour de nouveaux points. On dispose du paradigme training/test, apprentissage/évaluation.

     On peut toujours (essayer de) calibrer le modèle par validation croisée (en découpant le jeu de données). Cela revient à minimiser une **estimation** du risque théorique.
   - **En classification non supervisée, on ne connaît pas la « bonne réponse »**. On ne dispose plus du paradigme training/test, apprentissage/évaluation : autrement dit on ne peut pas aussi facilement estimer le risque théorique. On se rabat sur des heuristiques.

b) Les heuristiques en classification non supervisée

1. **La méthode du coude.** La fonction risque empirique $k \mapsto R(k)$ est typiquement concave, on cherche le point « coude ». Cette heuristique se justifie ainsi : lorsque le risque empirique décroît moins vite, on peut espérer que l'augmentation de k correspond à une assimilation moindre d'information et un apprentissage plus important du bruit. Il ne sert alors à rien de chercher un k plus important.
2. **La stabilité :** on regarde si plusieurs itérations de l'algorithme rendent le même genre de résultat. Si les barycentres changent du tout au tout, on a certainement un modèle insuffisamment robuste.
3. Et [...] de nombreuses autres idées, c'est un domaine de recherche entier.

IV Algorithme des k-moyennes vs algorithme glouton

1 Reformulation de l'algorithme

On a vu en exercice que la minimisation de $c \mapsto \sum_{i=1}^{n} ||X_i - c||^2$ s'assimile à un problème de minimisation qui a une unique solution, à savoir le barycentre.
Dans le cas des k-means, une fois k fixé, on cherche à la fois des centres $c_1, \cdots, c_k$ et une partition $C_1, \cdots, C_k$ qui minimise la quantité $\sum_{j=1}^{k}\sum_{X\in C_j} ||X - c_j||^2$.

2 Algorithme glouton

Une solution aurait été de produire un **algorithme glouton** :

1. Parcourir toutes les partitions possibles en k classes,
2. Pour chaque partition, calculer les barycentres $(c_1, \cdots, c_k)$ et $\sum_{j=1}^{k} \sum_{X \in C_j} ||X - c_j||^2$.
3. Retourner la partition qui minimise cette quantité.

On se rabat sur l'algorithme des k–moyennes qui termine plus vite. Cet algorithme a pour complexité le nombre de partitions, qui est exponentiel en la taille des données : inutilisable en pratique.
Cependant cet algorithme des k-moyennes ne garantit pas de produire la meilleure solution : au contraire, on obtient « souvent une bonne solution », un peu comme un « bon » minimum local (qu'on espère pas trop loin du minimum global recherché).

V Concepts à retenir

1. Apprentissage supervisé/ non supervisé
2. Le piège du surapprentissage : le modèle de plus grande complexité surapprend les données (overfitting, overlearning)
3. La minimisation du risque empirique favorise les modèles les plus complexes
4. Apprentissage supervisé : une méthode générique est la validation croisée (crossvalidation) ou la séparation en données d'apprentissage et données de sélection du modèle
5. Apprentissage non supervisé : heuristiques.

VI Exercices

Algorithme des k–moyennes – démonstrations & solutions

Exercice 1 Soit $m \in E$. On a $f(m) = f(\mu+(m-\mu)) = \frac{1}{n}\sum_{i=1}^{n}(||(x_i-\mu)+(\mu-m)||^2) = \frac{1}{n}\sum_{i=1}^{n}(||(x_i-\mu)||^2+2(\,x_i-\mu \mid \mu-m\,)+||(\mu-m)||^2)$

AInsi $f(m) = \frac{1}{n}\sum_{i=1}^{n}||(x_i-\mu)||^2 + \frac{2}{n}\sum_{i=1}^{n}(\,x_i-\mu \mid \mu-m\,) + \frac{1}{n}\sum_{i=1}^{n}||(\mu-m)||^2 = f(\mu) + 2(\,\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu) \mid \mu-m\,) + ||(\mu-m)||^2$.

Or on a $\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu) = \frac{1}{n}\sum_{i=1}^{n}x_i - \frac{1}{n}\sum_{i=1}^{n}\mu = \mu - \frac{1}{n}n\mu = 0_E$.

On a donc $f(m) = f(\mu) + ||m-\mu||^2$ d'où le résultat.

Exercice 2

```
#distance entre deux points, numpy

def distance (A,B):
    return sum((A-B)**2)



#trouver le point le plus proche:
def distance_carre(X1,X2):
    return np.sum((X1-X2)**2)

def barycentre(listeX):
     return sum(listeX)/len(listeX)

def moment(listeX):
    mu=barycentre(listeX)
    mmt=sum( (listeX-mu)**2 )
    return mmt

def indice_plus_proche(X, liste):
    m=np.inf
    for i in range(len(liste)):
        Y=liste[i]
        M=distance_carre(X,Y)
        if M<m:
            m=M
            ind_plus_proche=i
    return ind_plus_proche
#variante en utilisant à fond numpy:

def indice_plus_proche_bis(X, liste):
    liste_distances=[distance_carre(X, Y) for Y in liste ]
    print(liste_distances)
    return np.argmin(liste_distances)


def sont_égaux(A,B):
    return (A==B).all()
```

Exercice 3 Cela va dessiner trois nuages, chacun de 30 points, sur la même figure. L'un est de centre (0.5,1.5), l'un de centre (1.5,0.5) et l'un de centre (1.75, 1.75). Les trois nuages de points n'ont pas la même dispersion (le troisième a une variance 4 fois plus petite que les deux autres).

Exercice 4 1. Les liste $[a,b,c]$ et $[c,b,a]$ représentent le **même** ensemble de points, mais sont considérées comme differentes. Ainsi on va faire beaucoup trop de False : on a un problème de complexité lorsque l'on itère l'appel à la fonction `liste_egales`.

2. Solutions :

- Changer la représentation : la représentation par des listes pour des ensembles est inappropriée, il vaudrait mieux avoir des dictionnaires. Le problème dans cette solution est que les tableaux numpy ne sont pas hashables. Il vaudrait donc mieux garder tous les tableaux dans une même liste et manipuler en ensembles les **indices entiers** de ces tableaux. A vous de jouer...
- Plutôt améliorer la fonction de comparaison pour que deux listes distinctes de points égales à permutation près soient décidées égales. On peut alors, par exemple, trier les listes par ordre lexicographique.
  Attention il faut réimplémenter cet ordre sur les tableaux numpy.

Remarquez que cette solution est de coût négligeable ici puisque les listes sur lesquelles la fonction va s'appliquer sont de petites tailles : elles sont de longueur k où k est le nombre de centres.