Signaux, Sons et Images pour l'Informaticien: Description de sons

Diane Lingrand

Polytech SI3

2018 - 2019

Cepstre [kɛpstr]

- Motivation : en parole, séparer les formants
- Transformation :

$$C(\tau) = C(s(t)) = F^{-1}(\ln(|F(s(t))|))$$

spectre	cepstre
fréquence	quéfrence
filtrage	liftrage
phase	saphe
période	répiode

- convolution (temporel) - multiplication (fréquentiel) - addition (cepstral)
- Méthodes pour accélérer : LPCC et MFCC

LPCC : Linear Prediction Cepstral Coefficients

- pré-amplification
- fenêtrage Hamming
- analyse prédiction linéaire (LPC : Linear Prediction Coefficients) : $a_i, i \in [0\ p]$
- analyse cepstrale (calcul par récurrence coefficients cepstraux à partir des coefficients LPC) :

$$c_i = a_i + \sum_{k=1}^{i-1} \frac{k}{i} c_k a_{i-k} \quad \text{pour} \quad i \in [1\ p] \tag{1}$$

$$c_i = \sum_{k=i-p}^{i-1} \frac{k}{i} c_k a_{i-k} \quad \text{pour} \quad i \in [p\ N] \tag{2}$$

équivalent au cepstre complexe

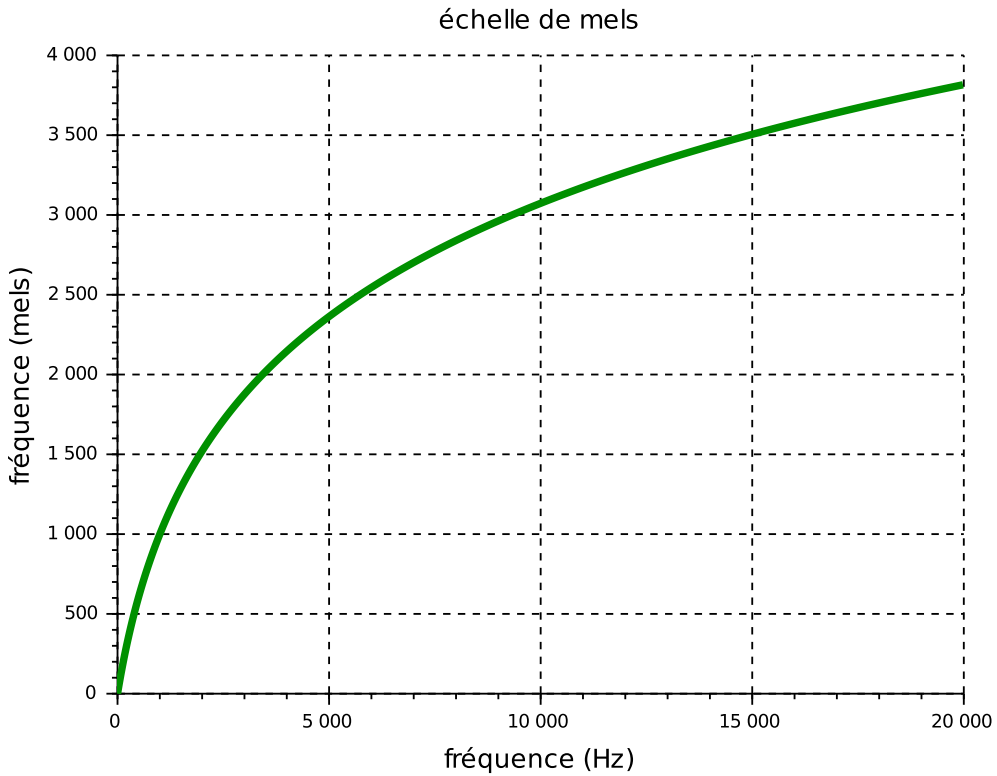
Echelle de mel

- Echelle basée sur la perception, créée en 1937.
- 1000 Hz = 1000 mels
- ratio identiques en mels équivalent à intervalles perçus identiques

$$m = 2595 \log_{10}(1 + \frac{f}{700})$$

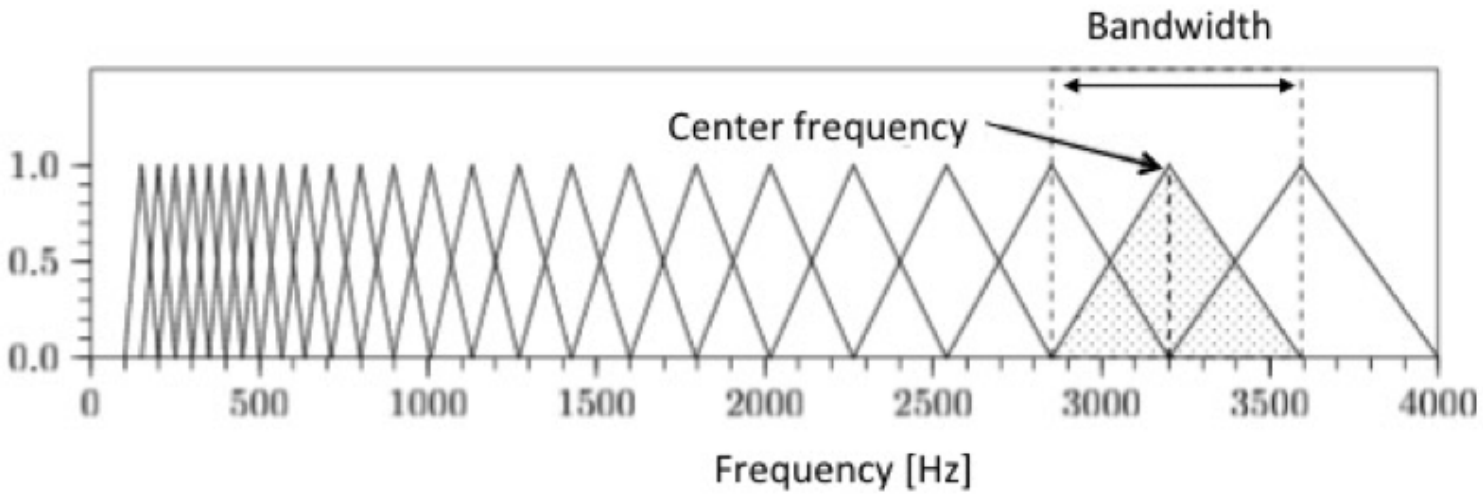
et réciproquement

$$f = 700\,(10^{\frac{m}{2595}} - 1)$$



MFCC : Mel-Frequency Cepstral Coefficients

- transformée de Fourier
- banc de filtres en triangles selon échelle de Mel
- transformée logarithmique
- transformée en cosinus discrète : les coefficients obtenus sont les MFCC



Pour le TP : calcul de coefficients MFCC avec aubio

- site web : `aubio.org`
- install ubuntu : sudo apt-get install aubio-tools
- utilisation : `aubiomfcc [[-i] source] [-r rate] [-B win] [-H hop] [-v] [-h]`
  - `-i source` : fichier audio
  - `-r rate` : re-échantillonnage du signal à la fréquence `rate` en Hertz entiers
  - `-B win` : taille de la fenêtre (par défaut 512)
  - `-H hop` : décalage entre deux fenêtres (par défaut 256)

Pour le TP : calcul de coefficients MFCC avec python

- installation : `pip3 install python_speech_features`
- utilisation :

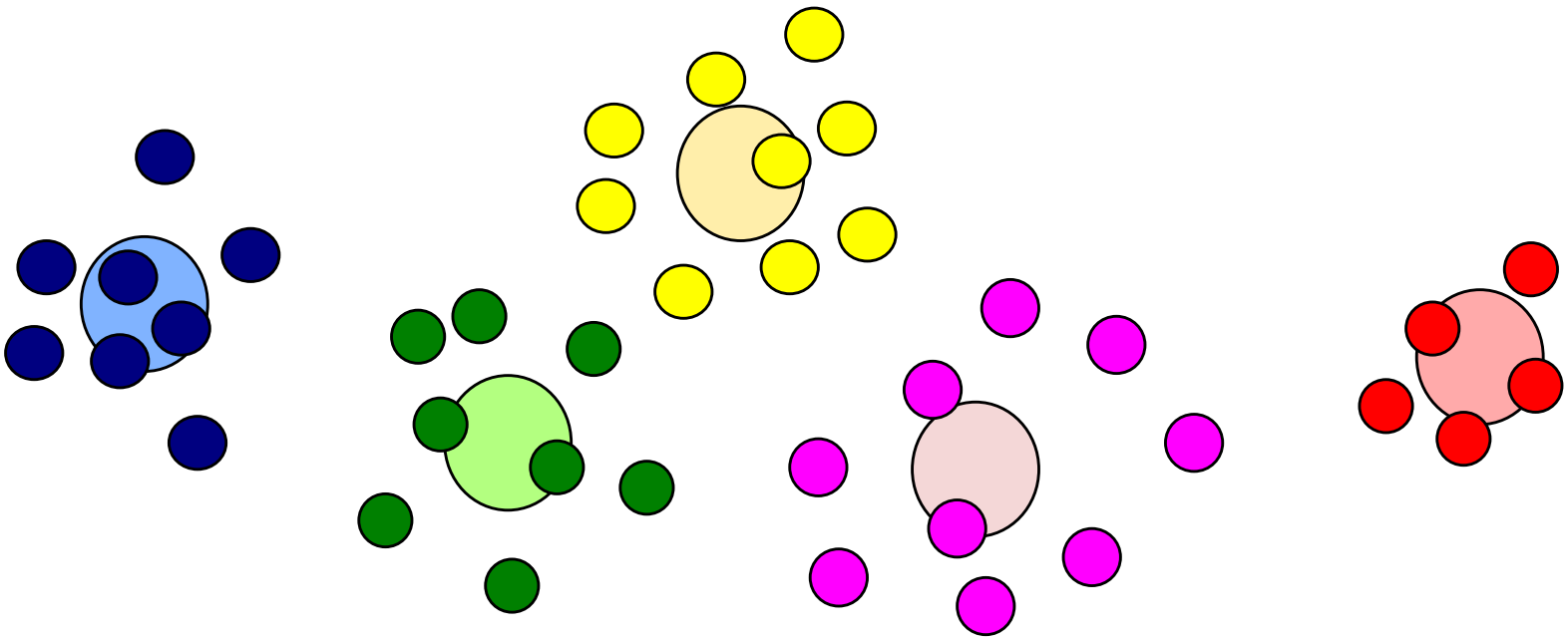
```
from python_speech_features import mfcc
import scipy.io.wavfile as wav

(rate,sig) = wav.read("piano.wav")
mfcc_feat = mfcc(sig,rate)
print(type(mfcc_feat)) #<class 'numpy.ndarray'>
print(mfcc_feat.shape) #(175, 13)
```

Pour le TP : classification par K-means, le principe

Partitionnement d'un ensemble de données x_i en K ensembles $E_1, E_2, .., E_k$ en minimisant la somme des distances des données aux barycentres des ensembles :

$$\min \sum_{i=1}^{k} \sum_{x \in E_i} \|x - b_i\|^2 \text{ avec } b_i \text{ barycentre de } E_i$$

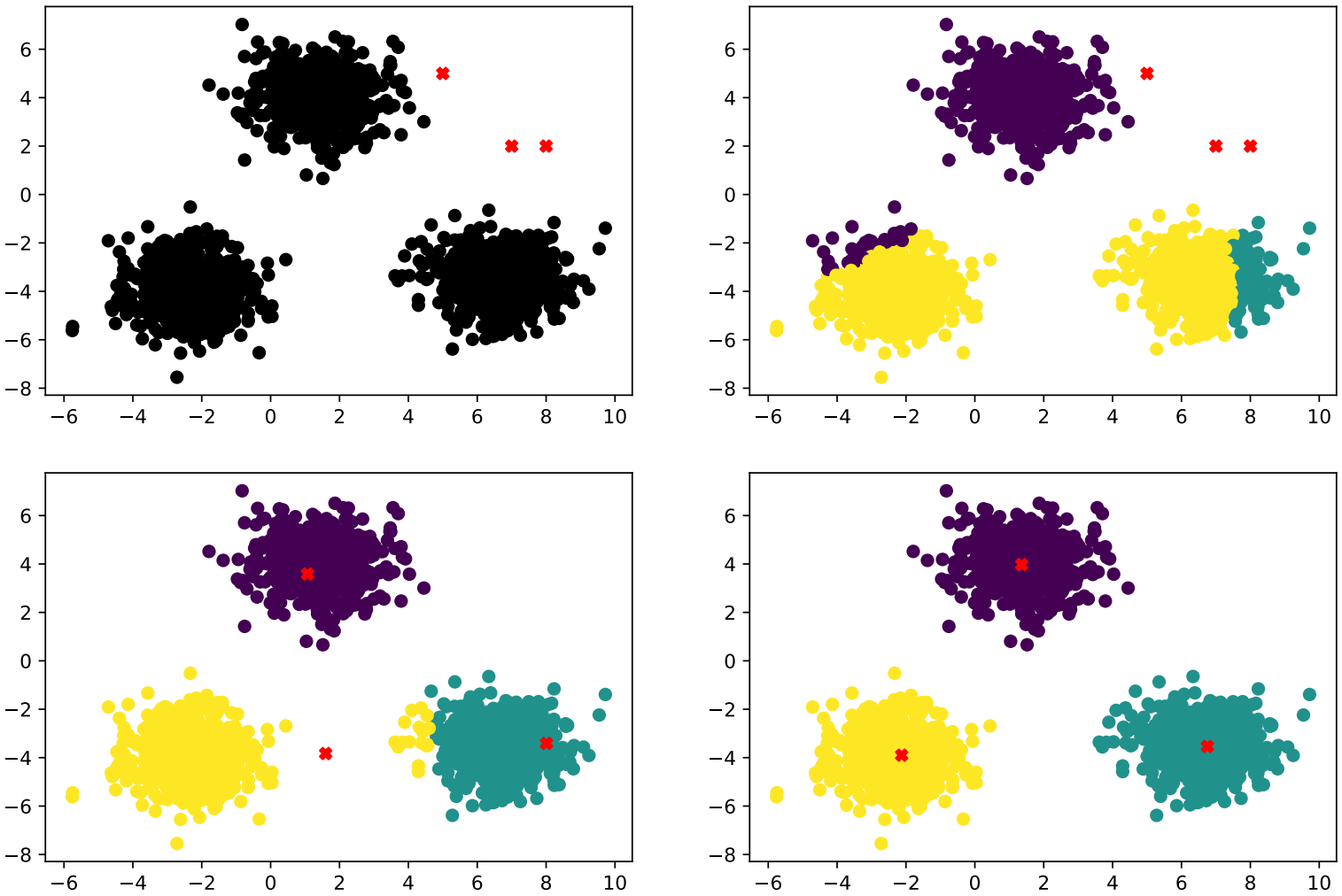


L'algorithme k-means

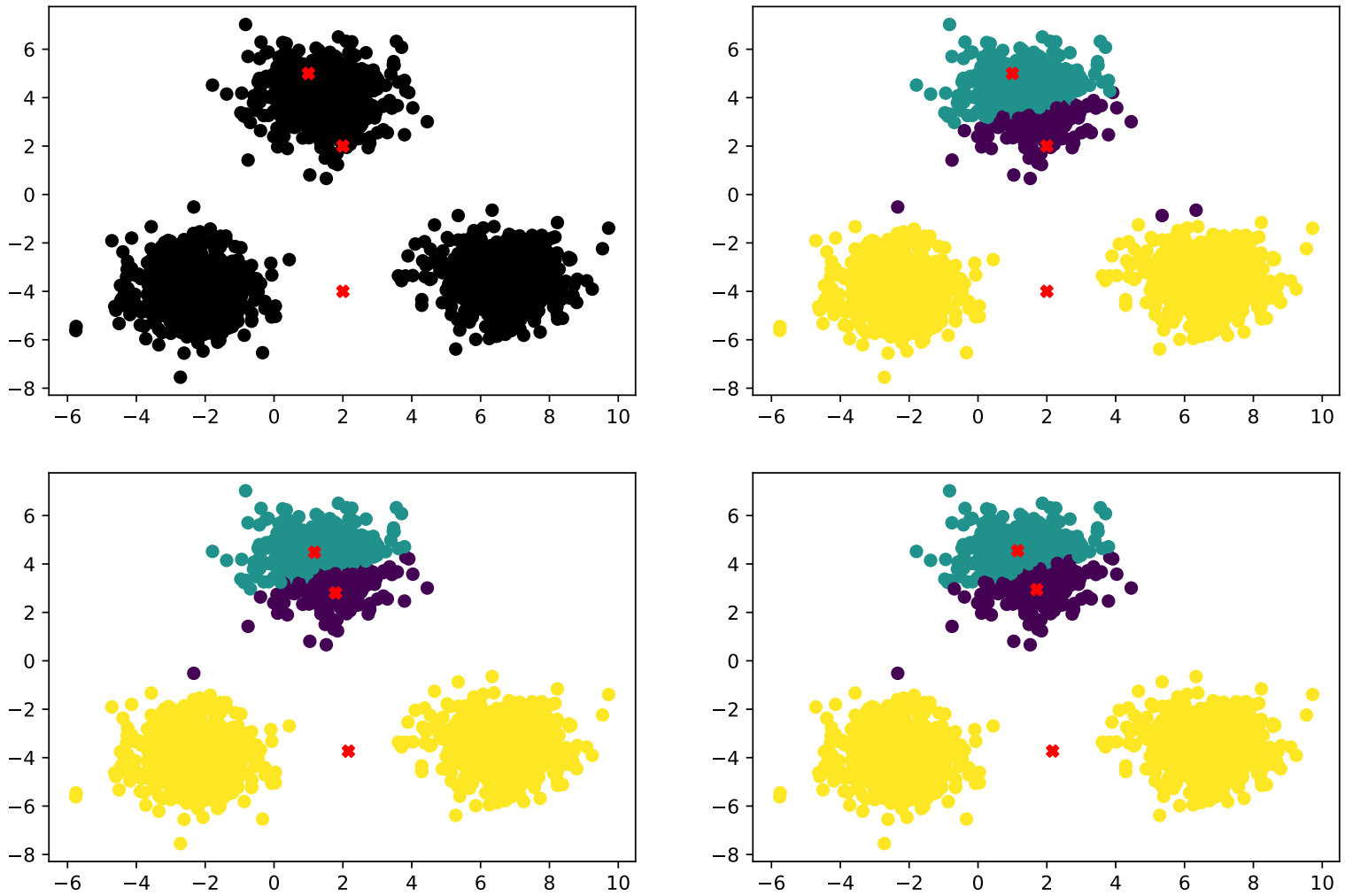
Ayant choisi k, l'algorithme se déroule en 4 étapes :

1. Initialiser la position des k barycentres et leur affecter l'étiquette k
2. Associer chaque object à la classe du barycentre le plus proche
3. Calculer les barycentres de chaque classe
4. Retour à l'étape 2 tant que les barycentres se déplacent

Exemple de k-means



Exemple de k-means : mauvaise initialisation

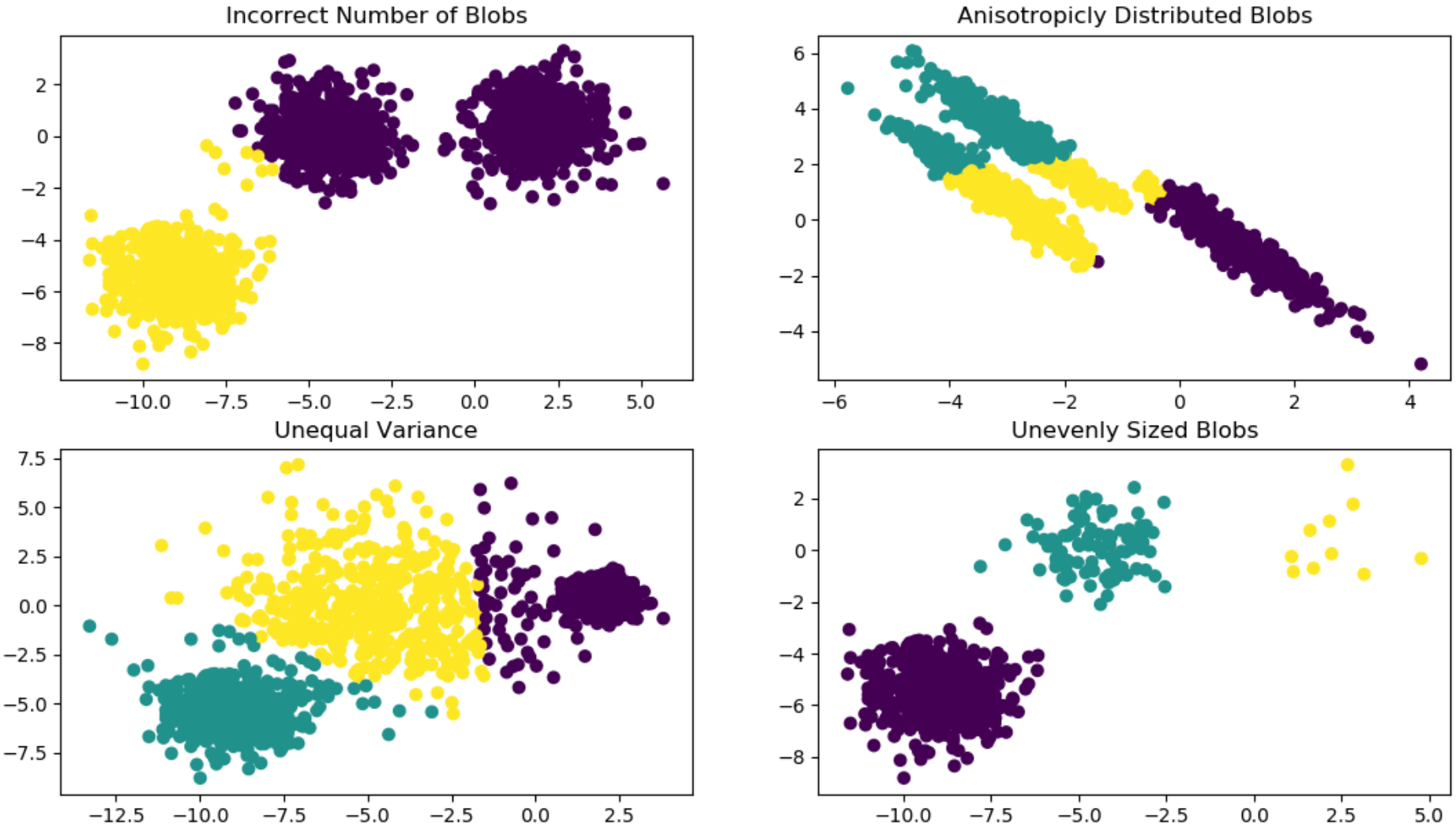


Force et faiblesses de k-means

- Force :
  - Relativement efficace
  - $O(tkn)$, où n nb. d'objects, k nb. de classes, et t nb. d'itérations. Habituellement, $k, t << n$.
- Faiblesses :
  - minimum local : essayer plusieurs initialisation (`kmeans++`)
  - nécessité de connaître le nombre de classes k
  - non adapté aux formes non convexes

Hypothèses

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html



Pour le TP : classification par K-means, en python

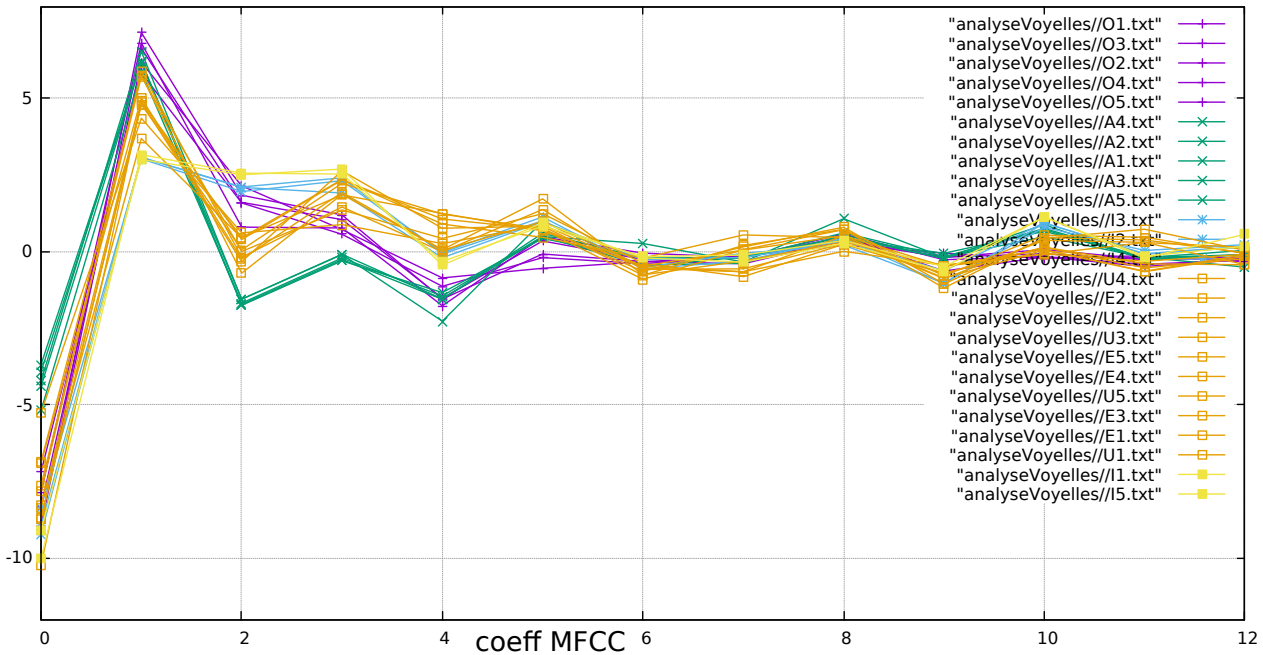
```
class sklearn.cluster.KMeans(n_clusters=8, init='k-means++',
n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto',
verbose=0, random_state=None, copy_x=True, n_jobs=1,
algorithm='auto')
```

Documentation sur http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

- init : 'k-means++', 'random' or an ndarray Method for initialization, defaults to `'k-means++'`
  - `'k-means++'` : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
  - `'random'`: choose k observations (rows) at random from data for the initial centroids.
  - If an `ndarray` is passed, it should be of shape `(n_clusters, n_features)` and gives the initial centers.
- `n_init : int, default:` 10. Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

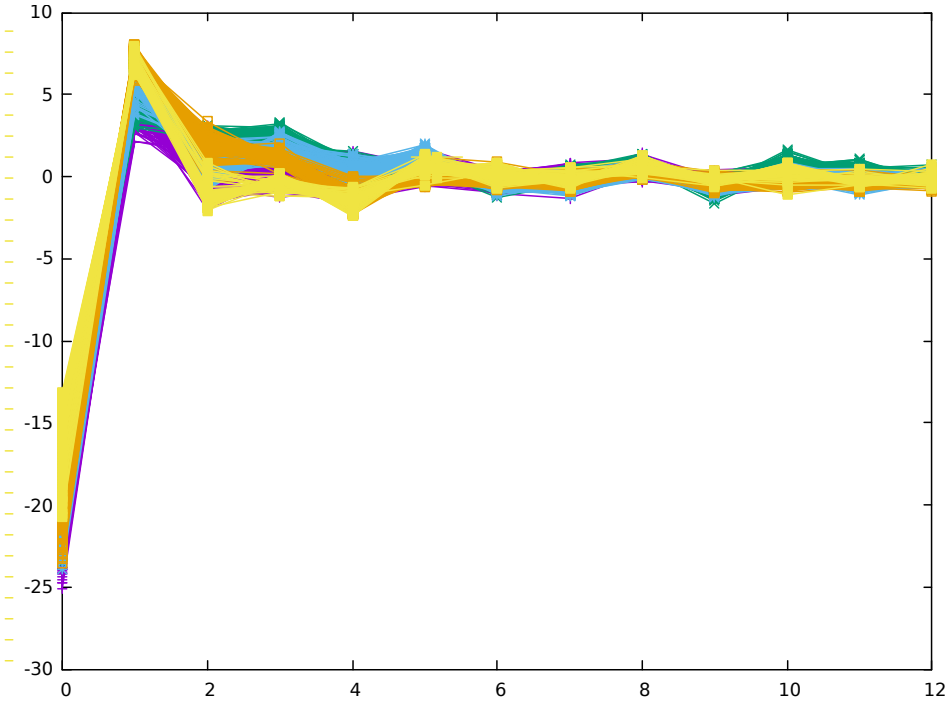
Pour le TP : exemple avec des voyelles

- 5 échantillons par voyelle 'A', 'E', 'I', 'O', 'U'.
- **1ère expérience :** 1 MFCC (13 valeurs) par fichier audio
- K-means avec 5 classes



Pour le TP : exemple avec des voyelles

- 5 échantillons par voyelle 'A', 'E', 'I', 'O', 'U'.
- **2ème expérience :** 1 MFCC par fenêtre de 512 échantillons, avec décalage de 256 : environ 40 MFCC par fichier audio
  - classification de tous les MFCC (k-means avec k=5) indépendemment des fichiers audio



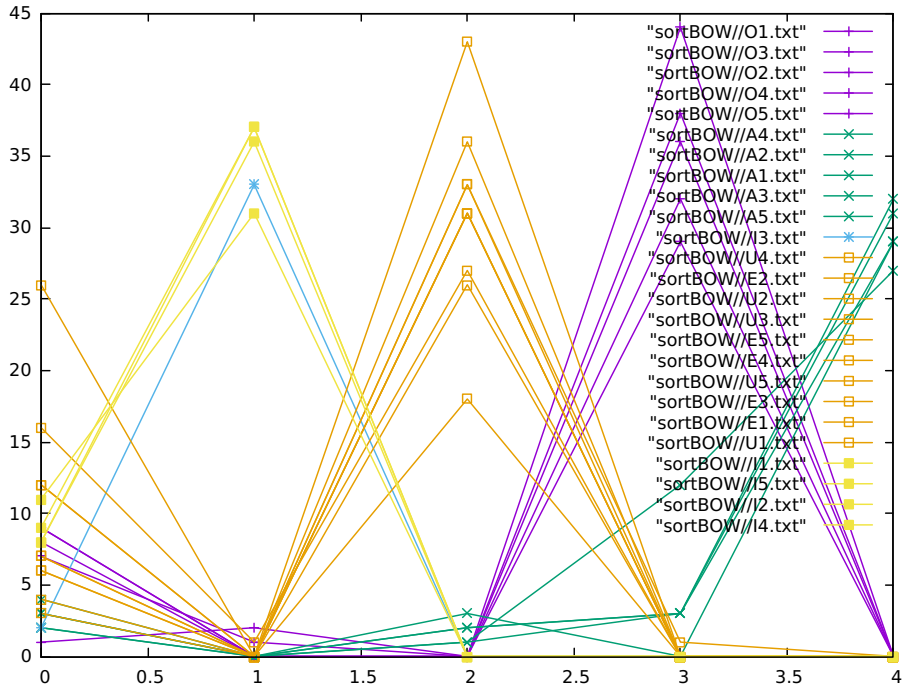
Pour le TP : exemple avec des voyelles

- 5 échantillons par voyelle 'A', 'E', 'I', 'O', 'U'.
- **2ème expérience :** 1 MFCC par fenêtre de 512 échantillons, avec décalage de 256 : environ 40 MFCC par fichier audio
  - pour chaque fichier audio, on regarde la classification de ses MFCC et on compte les occurrences de chaque classe : création d'un vecteur de classes (5 éléments). Classification des vecteurs obtenus

A1	3	0	2	3	31	E1	6	0	43	0	0
A2	2	0	3	0	29	E2	7	0	36	0	0
A3	3	0	2	3	29	E3	3	0	33	1	0
A4	2	0	1	3	32	E4	7	0	31	0	0
A5	4	0	1	12	27	E5	4	0	33	0	0
I1	9	37	0	0	0	O1	9	0	0	44	0
I2	8	37	0	0	0	O2	8	0	0	36	0
I3	2	33	0	0	0	O3	1	2	0	32	0
I4	8	36	0	0	0	O4	9	0	0	38	0
I5	11	31	0	0	0	O5	7	1	0	29	0
U1	12	0	31	0	0						
U2	6	0	31	0	0						
U3	26	0	18	0	0						
U4	16	1	27	0	0						
U5	12	0	26	0	0						

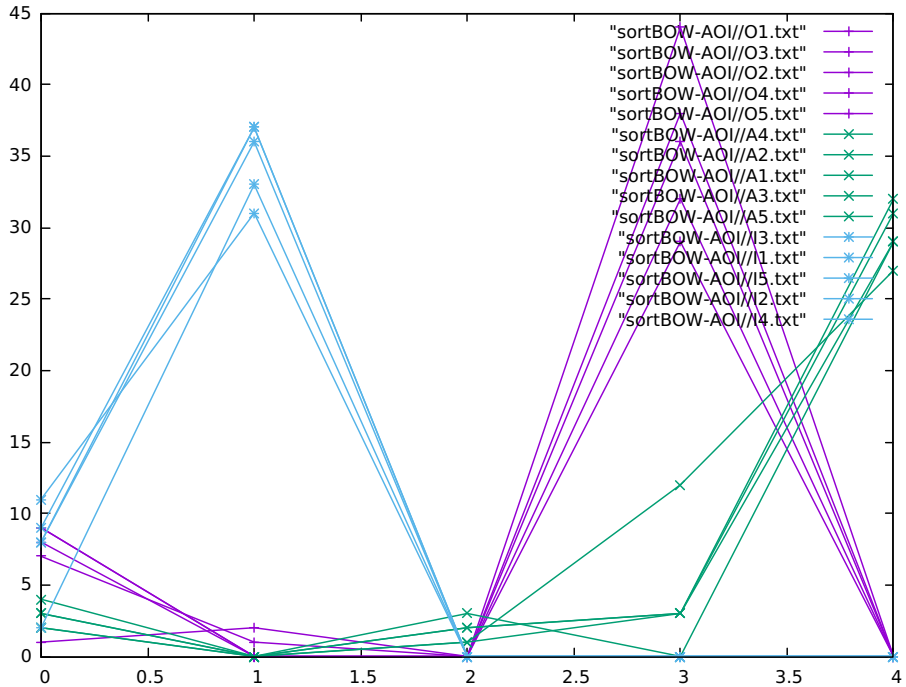
Pour le TP : exemple avec des voyelles

- 5 échantillons par voyelle 'A', 'E', 'I', 'O', 'U'.
- **2ème expérience :** 1 MFCC par fenêtre de 512 échantillons, avec décalage de 256 : environ 40 MFCC par fichier audio
  - 1 vecteur de votes par fichier audio. Classification de ces vecteurs



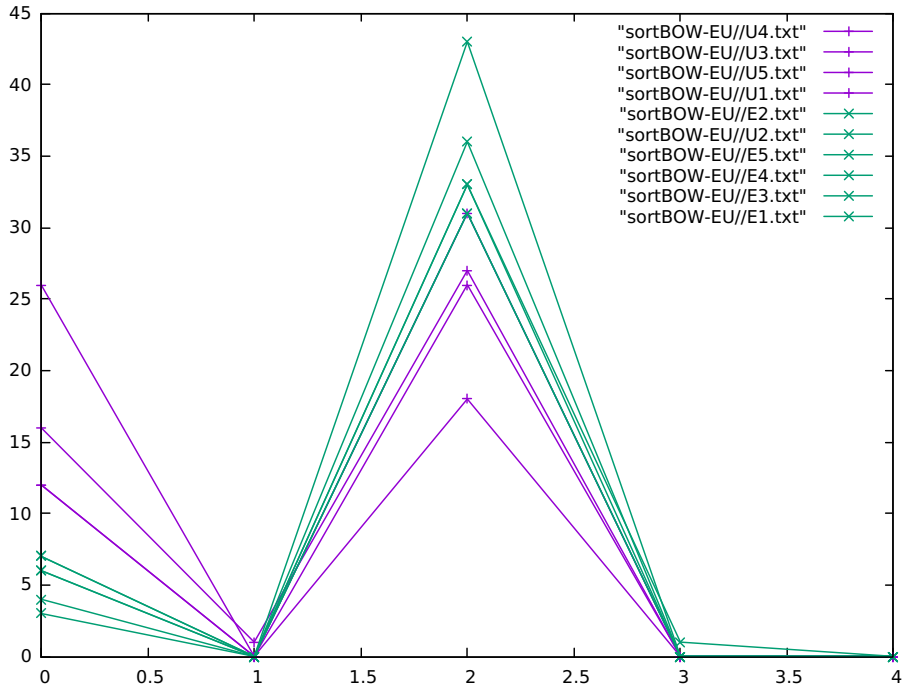
Pour le TP : exemple avec des voyelles

- 5 échantillons par voyelle 'A', 'E', 'I', 'O', 'U'.
- **2ème expérience :** 1 MFCC par fenêtre de 512 échantillons, avec décalage de 256 : environ 40 MFCC par fichier audio
  - 3 classes 'A', 'O', 'I'



Pour le TP : exemple avec des voyelles

- 5 échantillons par voyelle 'A', 'E', 'I', 'O', 'U'.
- **2ème expérience :** 1 MFCC par fenêtre de 512 échantillons, avec décalage de 256 : environ 40 MFCC par fichier audio
  - 2 classes 'E', 'U'

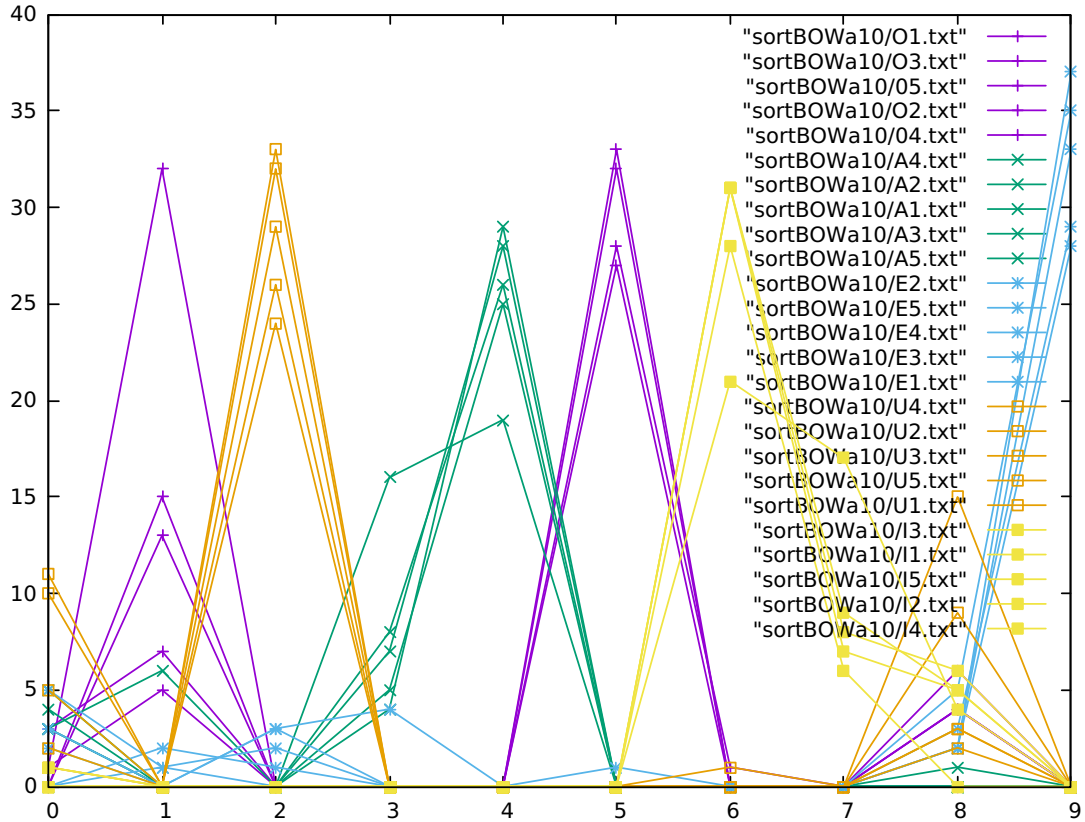


Pour le TP : exemple avec des voyelles BOW 10

On recommence avec les mêmes vecteurs de MFCC mais, lors de la 1ère classification par k-means, on choisit k=10. On décrit chaque son par les résultats de ses MFCC par cette classification :

A1	5	0	0	4	29	0	0	0	1	0
A2	4	0	0	5	25	0	0	0	0	0
A3	3	0	0	8	26	0	0	0	0	0
A4	3	0	0	7	28	0	0	0	0	0
A5	3	6	0	16	19	0	0	0	0	0
E1	3	0	3	4	0	0	0	0	2	37
E2	0	2	1	0	0	0	0	0	5	35
E3	0	1	0	0	0	1	0	0	2	33
E4	5	1	2	0	0	0	0	0	2	28
E5	2	0	3	0	0	0	0	0	3	29
I1	0	0	0	0	0	0	31	9	5	0
I2	0	0	0	0	0	0	31	8	6	0
I3	1	0	0	0	0	0	28	6	0	0
I4	1	0	0	0	0	0	31	7	5	0
I5	0	0	0	0	0	0	21	17	4	0
O1	0	15	0	0	0	32	0	0	6	0
O2	0	13	0	0	0	27	0	0	4	0
O3	1	5	0	0	0	28	1	0	0	0
04	3	7	0	0	0	33	0	0	4	0
05	1	32	0	0	0	0	0	0	4	0
U1	1	0	33	0	0	0	0	0	9	0
U2	2	0	32	0	0	0	0	0	3	0
U3	5	0	24	0	0	0	0	0	15	0
U4	11	0	29	0	0	0	1	0	3	0
U5	10	0	26	0	0	0	0	0	2	0

Et on classe les sons représentés par ces vecteurs de 10 éléments à nouveau par k-means (avec k=5) :



Conclusion sur les voyelles

Ce que l'on peut dire :

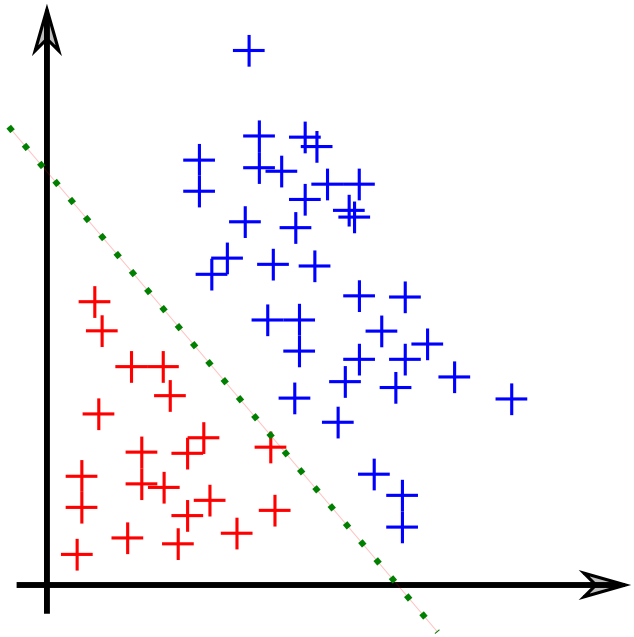
- Une première approche de classification mais
- les *bag of words(bow)* doivent être suffisamment grands (cf U4 et E2 si k=5)
- la méthode du k-means pour ce problème de classification n'est pas la plus pertinente : on n'a pas utilisé la connaissance de la classe
- l'évalution sur + de données reste à faire

Evaluation :

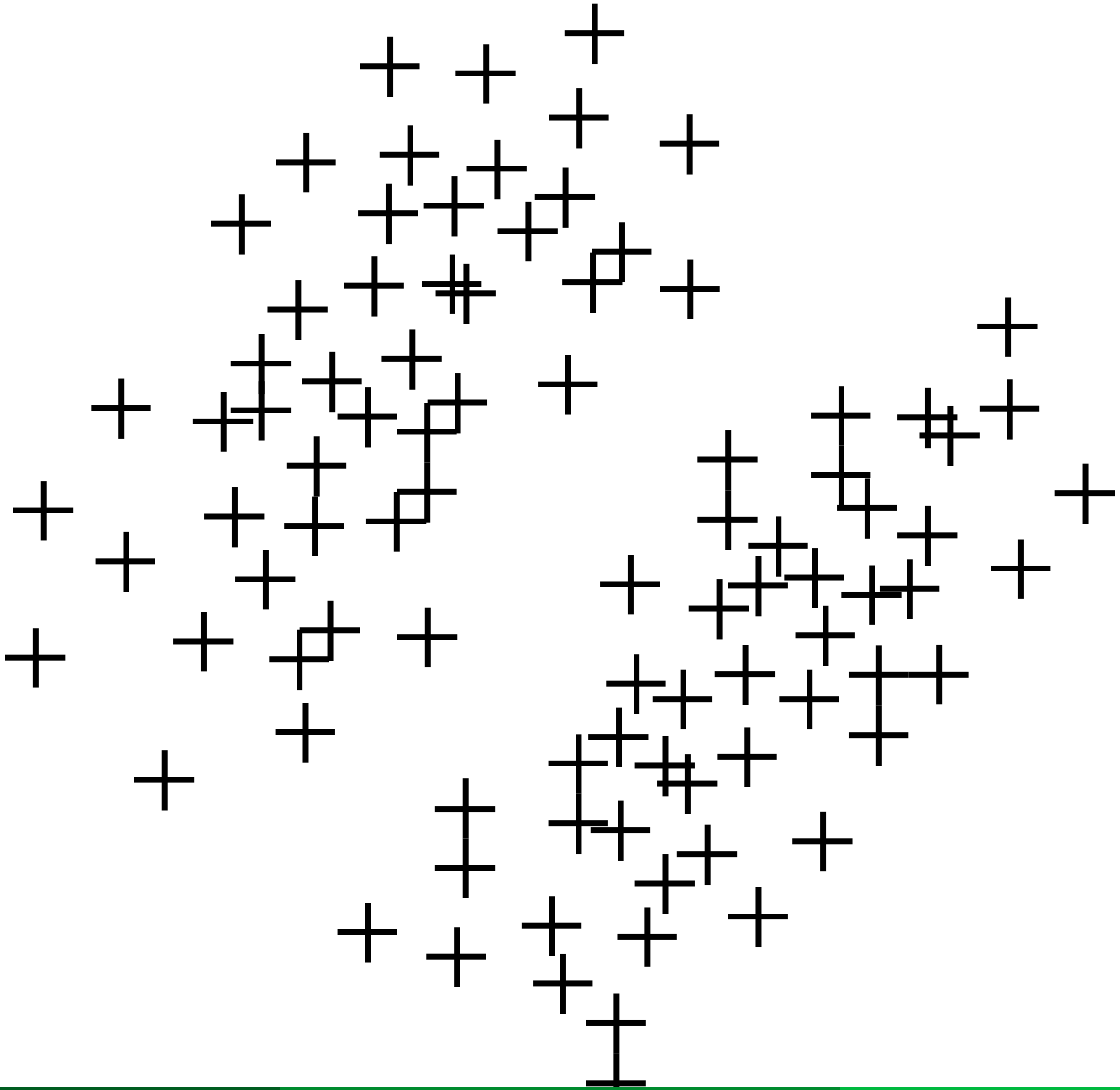
- On a appris les barycentres à l'aide d'un ensemble de données (5x5)
- Classer de nouvelles données à l'aide de ces barycentres
- Compter le nombre de données bien classées / mal classées

Apprentissage supervisé : regression logistique

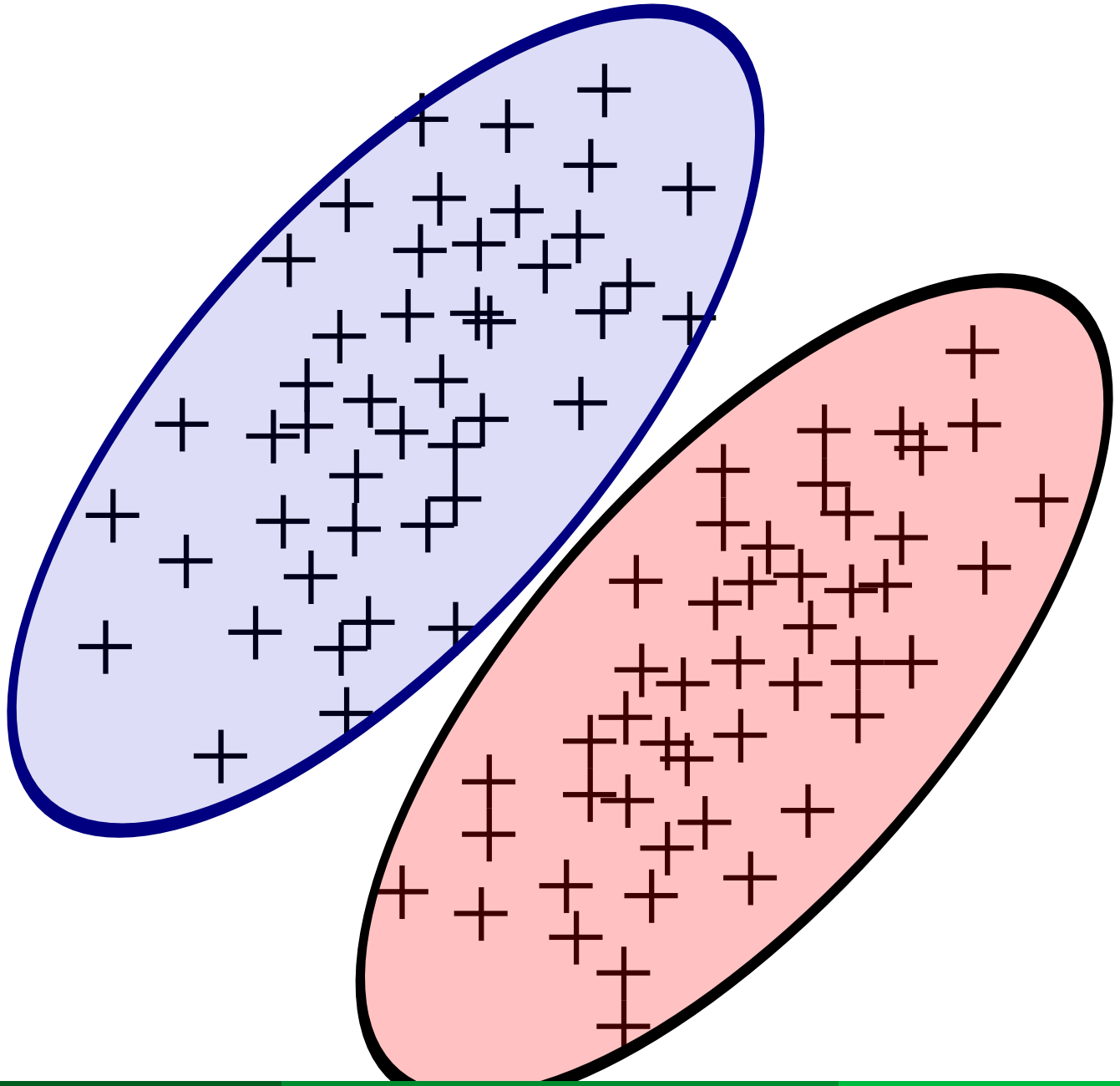
- Utilisation des labels des données
- Séparation de données selon leurs labels (0 ou 1).



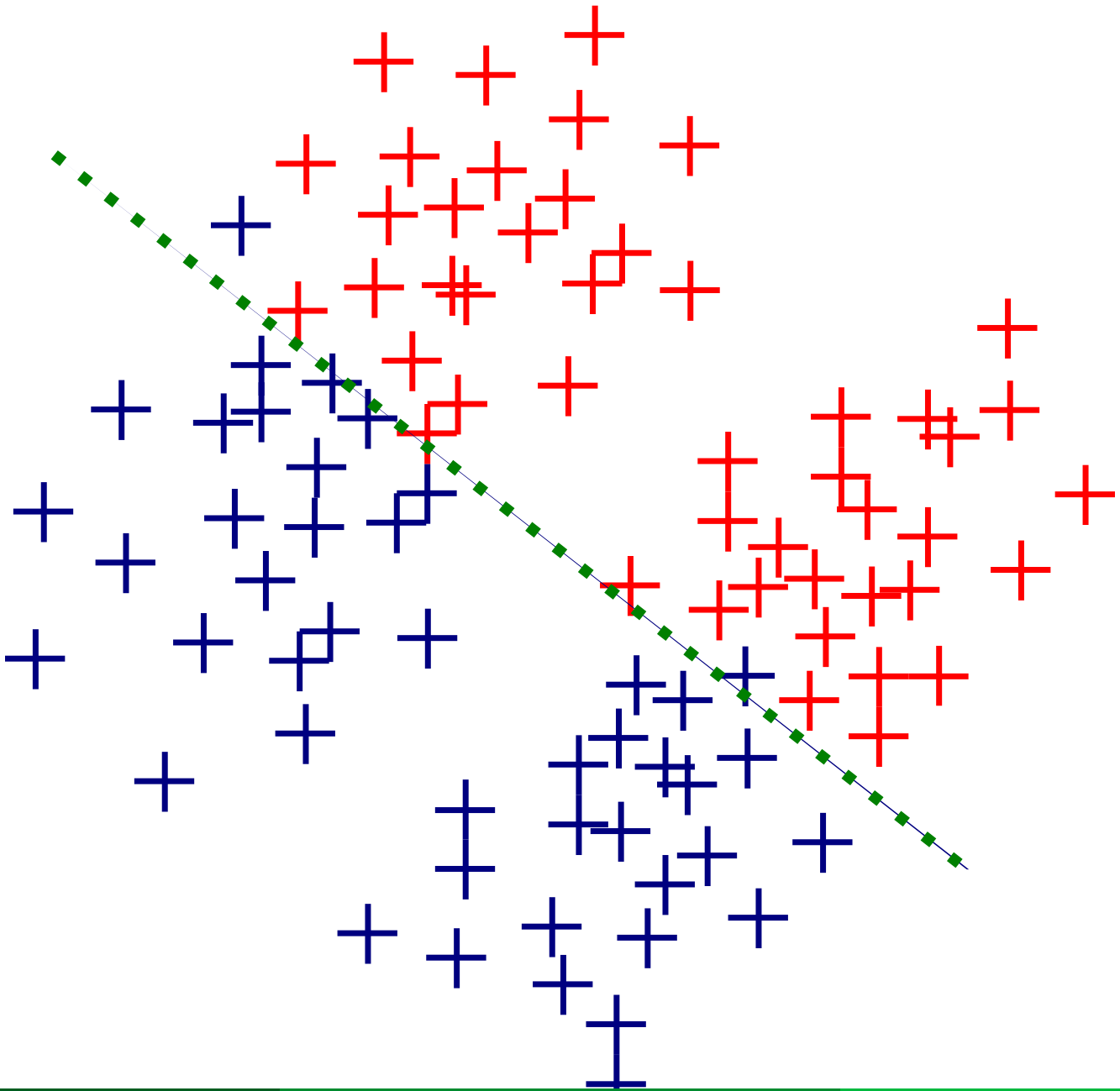
Retour sur *k-means* : points dans le plan



Séparation en deux ensembles par *kmean*



Utilisation des classes des points : régression logistique



Logistic regression en python

```
#création d'un objet de regression logistique
logisticRegr = LogisticRegression()
#apprentissage
logisticRegr.fit(bows, labels)
#calcul des labels prédits
labelsPredicted = logisticRegr.predict(bows)
#calcul et affichage du score
score = logisticRegr.score(bows, labels)
print("train score = ", score)
#sauvegarde de l'objet
with open('sauvegarde.logr', 'wb') as output:
   pickle.dump(logisticRegr, output, pickle.HIGHEST_PROTOCOL)
#chargement de l'objet
with open('sauvegarde.logr',  'rb') as input:
   logisticRegr = pickle.load(input)
```

A vous de jouer

- Choisir un problème :
  - reconnaître des syllabes différentes
  - reconnaître des voix
  - reconnaître des instruments de musique
  - reconnaître des styles de musique
  - ....
- Réaliser l'acquisition de données. Stocker les fichiers.
- Effectuer l'analyse
- Rédiger le tout et le rendre sous moodle (format pdf+sons).

Pour vous aider : script disponible sur moodle

```
import glob
from sys import argv
from python_speech_features import mfcc
from python_speech_features import logfbank
import scipy.io.wavfile as wav
from sklearn.cluster import KMeans
import numpy as np

# usage: python3 recosons2018.py k1 k2 verbose
# ATTENTION: les noms de fichiers ne doivent comporter ni - ni espace

#sur ligne de commande: les 2 parametres de k means puis un param de verbose
k1 = int(argv[1])
k2 = int(argv[2])

if argv[3] == "True":
    verbose = True;
else:
    verbose = False;

listSons=glob.glob("*.wav")

lesMfcc = np.empty(shape=(0, 13), dtype=float) # array of all MFCC from all sounds
dimSons = [] # nb of mfcc per file

for s in listSons:
    if verbose:
        print("###",s,"###")
    (rate,sig) = wav.read(s)
    mfcc_feat = mfcc(sig,rate)
    if verbose:
        print("MFCC: ", mfcc_feat.shape)
    dimSons.append(mfcc_feat.shape[0])
    lesMfcc = np.append(lesMfcc,mfcc_feat,axis=0)
```

Pour vous aider : script disponible sur moodle (2)

```
 #BOW initialization
 bows = np.empty(shape=(0,k1),dtype=int)

# everything ready for the 1st k-means
kmeans1 = KMeans(n_clusters=k1, random_state=0).fit(lesMfcc)
if verbose:
    print("result of kmeans 1", kmeans1.labels_)

#writing the BOWs for second k-means
i = 0
for nb in dimSons: # for each sound (file)
    tmpBow = [0]*k1
    j = 0
    while j < nb: # for each MFCC of this sound (file)
        tmpBow[kmeans1.labels_[i]] += 1
        j+=1
        i+=1
    copyBow = tmpBow.copy()
    bows = np.append(bows, [copyBow], 0)
if verbose:
    print("nb of MFCC vectors per file : ", dimSons)
    print("BOWs : ", bows)

#ready for second k-means
kmeans2 = KMeans(n_clusters=k2, random_state=0).fit(bows)
 if verbose:
     print("result of kmeans 2", kmeans2.labels_)
```