0.00.5em

0.0.00.5em

0.0.0.00.5em

0em

K_means Documentation

Release 1.0

B. Pacreau, G. Soulié

October 21, 2015

L'ensemble des sources de ce projet est disponible sur github :

Sous linux, vous pouvez facilement l'importer en tapant :

```
mkdir kmeans/
cd kmeans
git init
git remote http://github.com/pikkendorff/kmeans master
```

L'architecture générale du projet est la suivante :

```
./kmeans/
      |--doc/
      |--input/
      |    |--input.csv
      |    |--bretagne.jpg
      |    |--.iris.csv
      |--output/
      |    |--affectation.csv
      |    |--centroids.csv
      |--kmeans.py
      |--es.py
      |--genData.py
      |--Observation.py
```

Lorsqu'on lance kmeans, les données présentes dans le fichier *./input/input.csv* sont étudiées, et le résultat est écrit dans deux fichiers, *./output/affectation.csv* et *./output/centroids.csv*

Les fichiers *{genAleatoire*, *genIris* et *genPicture}.py* vous permettent de générer un fichier input.csv corrsespondant respectivement aux sections *Données aléatoires*, *Données Iris* et *La Bretagne vue du ciel*. Bien évidemment, vous pouvez utiliser l'algorithme avec un autre jeu de données de votre choix, pourvu que le fichier *input.csv* respecte le bon format (cf *Format des données*).

CHAPTER

ONE

TABLE DES MATIÈRES

1.1 Introduction

Selon Wikipédia, “Le partitionnement en k-moyennes (ou k-means en anglais) est une méthode de partitionnement de données et un problème d’optimisation combinatoire”. Ceci étant dit, prenons le temps de présenter de manière plus général le contexte dans lequel cet algorithme est généralement utilisé : l’apprentissage automatique

1.1.1 Apprentissage Automatique (*Machine learning*)

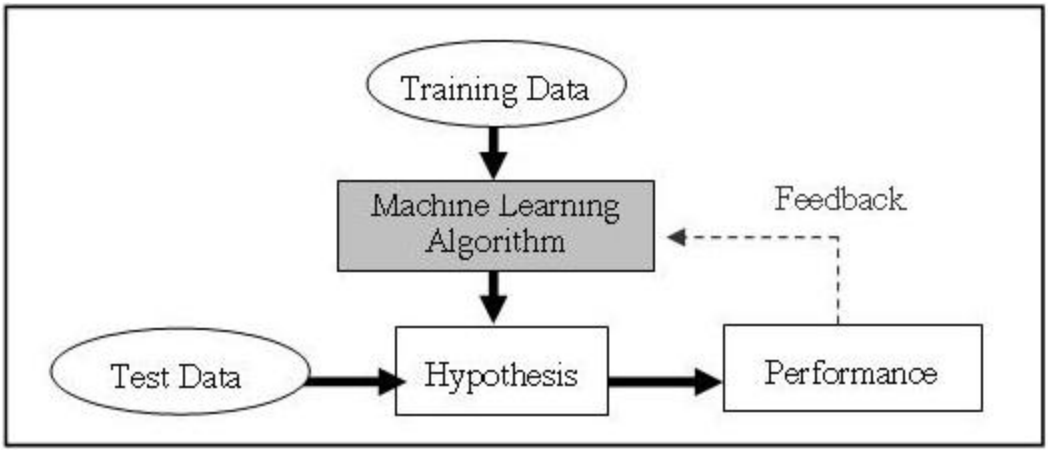
L’apprentissage automatique, plus connu sous le nom de *machine learning* en Anglais, est un sous-domaine de l’intelligence artificielle qui vise à effectuer des actions pour lesquelles ils n’ont pas été explicitement programés. Concrétement, on distingue l’apprentissage supervisé de l’apprentissage non supervisé.

- Apprentissage supervisé (*supervised learning*) : Consiste à attribuer une étiquette (*label*) à des données.
- Apprentissage non supervisé (*unsupervised learning*) : Consiste à séparer un ensemble de données en plusieurs catégories

1.1.2 Apprentissage supervisé

C’est le cas par exemple des célébres de classification d’image du type reconnaissance de chiffres manuscrit (MNIST), d’images(ImageNet). C’est également le cas si l’on veut determiner le prix d’un appartement en fonction de ses caractéristiques, ou encore la gravité d’une tumeur.

Le principe de fonctionnement est simple : on utilise un grand ensemble de données déjà étiquetée (appelé ensemble d’entrainement) pour construire un modèle, qui nous servira ensuite a estimer l’étiquette d’une donnée non étiquetée. Un jeu de données étiquetées (appelé ensemble de test), non utilisées pour l’entrainement du réseau, nous sert à vérifier la validité du modèle :



Parmi les grands algorithmes d'apprentissage supervisé, on compte les réseaux de neurones.

1.1.3 Apprentissage non supervisé

L'exemple classique pour évoquer l'apprentissage non supervisé concerne la segmentation du marché d'une entreprise. Une entreprise possède un certains nombre de clients et aimerait les dissocier en plusieurs catégories, mais ne connait pas forcément le nombre, la taille ou le type de catégorie à l'avance. Il s'agit de trouver un 'juste milieu' entre prendre une catégorie par individu, et une pour tous.

Un autre exemple que nous proposons, est celui d'une compagnie des Telecom qui voudrait installer un réseau d'antennes en Bretagne. Ne disposant que d'un nombre limité d'antennes, la compagnie souhaite maximiser l'emplacement de celles-ci. Une solution pour elle est de faire appel à un algorithme d'apprentissage non supervisé pour séparer la population en différentes catégories, et ensuite disposer une antenne pour chaque catégorie. Il s'agit donc de détecter une structure cachée dans l'ensemble de données.

L'un des algorithmes les plus célèbres pour faire de l'apprentissage non supervisé est précisemment l'algorithme des k-moyennes, plus connus sous son nom anglais *k-means*.

En particulier, nous allons étudier dans ce projet le deuxième exemple proposé : utiliser k-means pour positionner des antennes téléphoniques en Bretagne.

1.2 Etude théorique de k-means

1.3 Implémentation de k-means

Pour implémenter l'algorithme kmeans, nous avons utilisé deux fichiers : kmeans.py et Observation.py

Le fichier kmeans contient la focntion compute_kmeans

1.3.1 La fonction compute_kmeans

kmeans.**compute_kmeans**(*k*, *display=False*, *max_iteration=99999*)

Compute and display the k-means algorithm on the input file (*./input/input.csv*)

Parameters

- **k** (*int*) – the k of k-means : number of centroids

- **max_iteration** (*int*) – the number maximum of iteration we allow
- **display** (*boolean*) – if True, the first and the second coordinate of the populations are displayed setep by step

compute_kmeans fait appel au fichier es.py qui regroupe l'ensemble des opérations d'écriture, de lecture et d'affichages des *données*

```
population = es.read_kmeans_input()
dimension = len(population[0].values)
```

Phase 1 - Initialisation

```
#centroids initialisation:
centroids=[]
isSelected=[]
for i in range(len(population)):
        isSelected.append(0)
for i in range(k):
        while True:

                #centroids are ranomly choose in the population
                index = int(floor(random.random()*len(population)))

                #We checked that we don't take the same centroid twice
                if isSelected[index]==0:
                        centroids.append(population[index].copy())
                        isSelected[index]=1
                        break

#affectation initialisation:
affectation=[]
for i in range(len(population)):
        affectation.append(0)
if display:
        es.display(population,None,"Population : ")

#Loop stop condition initialisation:
stop=False


iteration = 0
while not stop and iteration < max_iteration:
        iteration+=1
```

Phase 2 - Affectation

```
#if display, we print the population and the centroids
if display:
        es.display(population,centroids,"computing k-means : \
                iteration "+str(iteration))

#Compute the distance between each observation and each centroid
distance=[[]]
for i in range(len(population)):
        distance.append([])
        for j in range(k):
                distance[i].append(population[i].dist(centroids[j]))
```

```
    #The loop stop condition is fixed to True
    stop = True

    #Affect the nearest centroid to each observation.
    for i in range(len(population)):
            index_du_minimum = distance[i].index(min(distance[i]))
            if not affectation[i]==index_du_minimum:
                    affectation[i]=index_du_minimum

    #If there is any changement, the loop stop condition became false
                    stop = False
```

Phase 3 - Calculation

```
    #Compute the new centroids
    for j in range(k):
            centroid = Observation(dimension,type="centroid")
            for i in range(len(population)):
                    if affectation[i]==j:
                            centroid.add(population[i])
            centroids[j]=centroid
#write the output files
es.write_kmeans_output(population,centroids,affectation)

#if display, we print the population and the centroids
if display:
        es.display(population,centroids,"K-means computed")
```

1.3.2 La classe Observation

class kmeans.**Observation**(*dimension*)

Représente une observation

__init__(*dimension*)

Create a new observation.

Parameters **dimension** (*int*) – dimension of the observation

add(*observation*)

Add another observation values to self.values, with ponderation by self.weight. Typically use wen this instanciation of Observation is a centroid. This function permits to compute easily a barycentre of several observations

Parameters **observation** (Observation) – the observation to add to the barycentre

copy()

Return another obseration with the same values

Return type *Observation*

dist(*observation*)

compute and return the euclidian distance between self and another observation

Parameters **observation** (Observation) – the observation to compute the distance with

Return type float

set_values (*values*)
set the values of the observation

Parameters **values** (*float[]*) – valeurs de l'observation

Return type void

1.4 Traitement des données

1.4.1 Lire, écrire et afficher des données

Le fichier *es.py* regroupe l'ensemble des méthodes nécessaires à la lecture, à l'écriture et à l'affichage des données.

es.**display** (*population*, *centroids*, *title*, *keep*)
This fonction is used to display the population and the centroids on a graph

Parameters

- **population** (*Observation[]*) – set of observations
- **centroids** (*Observation[]*) – set of centroids
- **titre** (*String*) – name of the graphic

:arg keep : if true, the figure stay open until the user close it :type keep : boolean :rtype: void

es.**read_data** (*filename*, *skip_first_line=False*, *ignore_first_column=False*)
Loads data from a csv file and returns the corresponding list. All data are expected to be floats, except in the first column.

Parameters

- **filename** (*String*) – csv file name.
- **skip_first_line** (*boolean*) – if True, the first line is not read. Default value: False.
- **ignore_first_column** (*boolean*) – if True, the first column is ignored. Default value: False.

Returns a list of lists, each list being a row in the data file. Rows are returned in the same order as in the file. They contains floats, except for the 1st element which is a string when the first column is not ignored.

Return type float[][]

es.**read_kmeans_input** ()
This fonction reads the file "input.csv" within the input folder

Returns population based on the input file

Return type Observation[]

es.**write_data** (*data*, *filename*)
Writes data in a csv file.

Parameters

- **data** (*float[][]*) – a list of lists to write
- **filename** (*String*) – the path of the file in which data is written. The file is created if necessary; if it exists, it is overwritten.

es.**write_kmeans_input** (*population*, *dimension*)
This function write the file *input.csv* in the input folder

Parameters **population** (*float[][]*) – the set of observation we want to compute

es.**write_kmeans_output**(*population*, *centroids*, *affectation*)

This fonction writes the files *affectation.csv* and *centroids.csv* in the output folder.

Parameters

- **population** (*Observation[]*) – set of observations
- **centroids** (*Observation[]*) – set of centroids
- **affectation** (*int[]*) – list of the observation's centroid label

Return type void

1.4.2 Format des données

Les formats des fichiers d'entrée et de dortie *.csv* sont les suivants :

Le fichier des données d'entrée

./input/input.csv doit respecter le format suivant :

```
# no_observation,attribut_1,attribut_2,...,attribut_p
1,0.1,0.8,0.4,0.3
2,0.3,0.7,0.5,0.2
3,0.9,0.6,0.7,0.1
4,0.4,0.2,0.8,0.3
```

Le fichier des données affectées

./output/affectation.csv respecte le format suivant :

```
# no_observation,attribut_1,attribut_2,...,attribut_p,no_classe
1,0.1,0.8,0.4,0.3,3
2,0.3,0.7,0.5,0.2,2
3,0.9,0.6,0.7,0.1,1
4,0.4,0.2,0.8,0.3,3
```

Le fichier des centres

./output/centroid.csv respecte le format suivant :

```
# no_centre,attribut_1,attribut_2,...,attribut_p
1,0.2,0.3,0.4,0.5
2,0.3,0.2,0.6,0.4
3,0.1,0.9,0.5,0.3
```

1.5 Générations aléatoires de données

Le fichier *genData.py* permet de générer des gaussiennes en dimension n, grâce à la commande suivante :

```
python genData.py -type random
```

La méthode utilisée est alors gen_random_data, qui créé les gaussiennes

genData.**gen_random_data**(*gaussiennes*, *dimension=2*, *taille_population=1000*)

Génére des gaussiennes de dimension dimension, et de forme définies par l'argument gaussienne.

Parameters

- **gaussiennes** (*dictionnaire*) – dictionnaire des gaussiennes à générées, de la forme : {"gaussienne1":{"direction":[],centre[]}}
- **dimension** (*int*) – dimensions des observations

Return type void

Plusieurs configurations de gaussiennes en 2 dimensions sont proposées en exemple. Il est possible de changer l'exemple générer en utilisant l'option -s :

```
python genData.py -type random -s 1
```

Il est également possible d'afficher les deux premières coordonnées des données générées avec l'option -d :

```
python genData.py -type random -s 1 -d True
```

1.6 Les données Iris

Le fichier *genData.py* permet de générer les données iris, grâce à la commande suivante :

```
python genData.py -type iris
```

La méthode utilisée est alors gen_iris_data, qui récupére en fait le fichier iris.csv déjà créé.

genData.**gen_iris_data**()
Génére les données de type iris

Return type void

Il est également possible d'afficher les deux premières coordonnées des données générées avec l'option -d :

```
python genData.py -type iris -d True
```

1.7 La Bretagne vue du ciel !

Le fichier *genData.py* permet de générer des données à partir une image satellite de nuit. Il faut pour cela placer l'image dans le dossier input et utiliser la commande suivante :

```
python genData.py -type picture -name nom_de_la_photo
```

La méthode utilisée est alors gen_picture_data.

genData.**gen_picture_data**(*name*)
Génére les données en provenance d'une image placée dans le fichier input.

Parameters **name** (*String*) – nom de l'image à analyser.

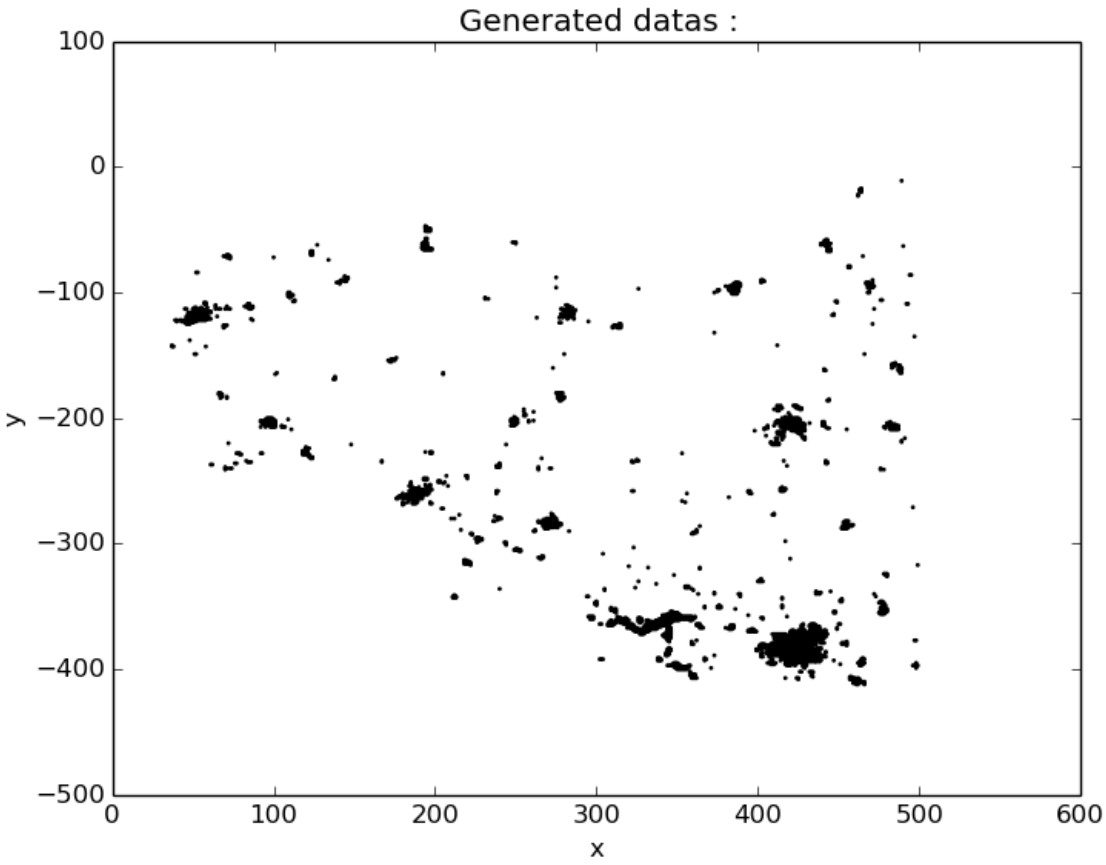
Return type void

Il est également possible d'afficher les données générées avec l'option -d :

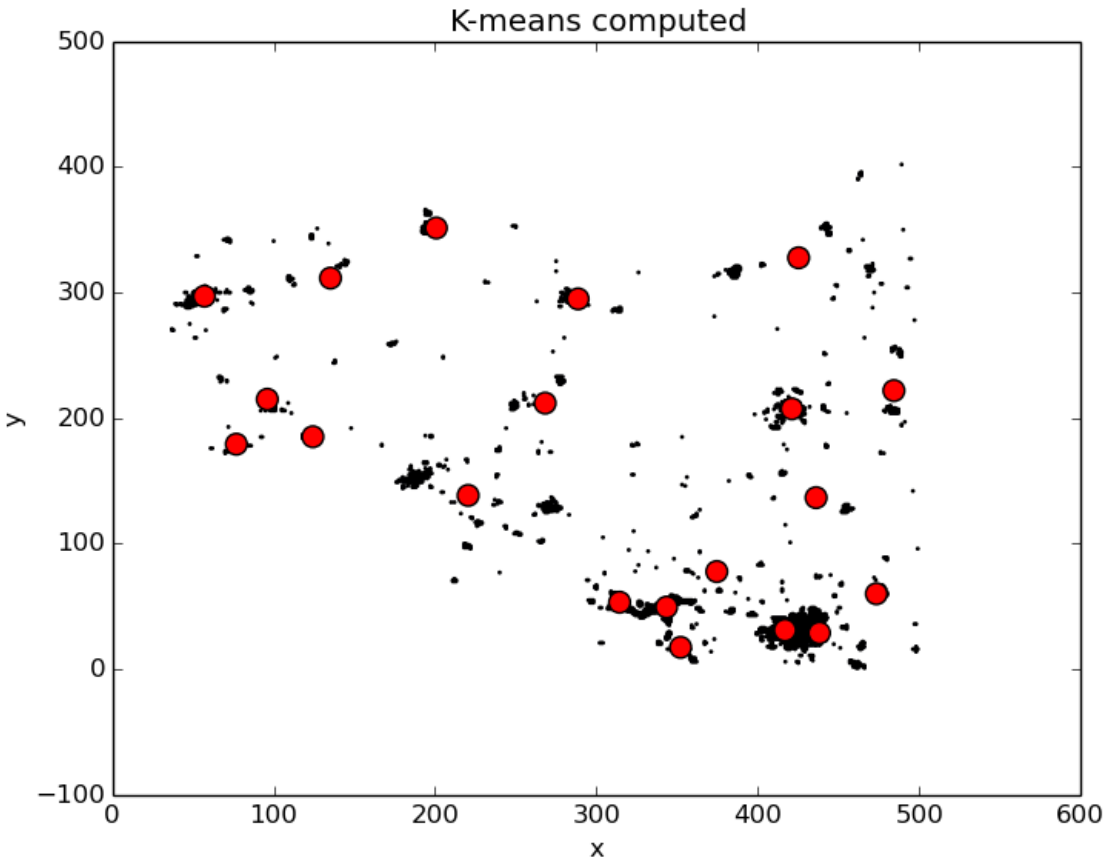
```
python genData.py -type picture -name bretagne.jpg -d True
```

Par défault, si l'option -name n'est pas utilisé, l'image bretagne.jpg est utilisé. Voivi l'image originale :

Voici ensuite les données générées grâce à cette image :



Et enfin voici le résultat de kmeans sur ces images :



- genindex
- modindex
- search

e

k