Algorithme des k plus proches voisins, un algorithme d'apprentissage automatique

L'algorithme des k plus proches voisins (en anglais « k-NN », diminutif de « k Nearest Neighbors ») est un algorithme utilisé, entre autre, dans le domaine de l'intelligence artificielle comme méthode d'apprentissage supervisée.

1. Pré-requis, objectifs, liens avec les programmes

Pré-requis

- Algorithmes de tri, tris par insertion, par sélection
- Coût linéaire et coût quadratique
- Python : importation d'une table depuis un fichier texte tabulé ou un fichier CSV
- Utilisation de bibliothèques Python

Objectifs

- Sensibiliser les élèves aux principes de l'intelligence artificielle
- Écrire l'Algorithme des k plus proches voisins
- Réinvestir les algorithmes de tri
- Réinvestir la notion de coût linéaire et de coût quadratique

Référence aux programmes de NSI

Programme de première NSI

- **Contenus** : Algorithme des k plus proches voisins
- **Capacités attendues** : Écrire un algorithme qui prédit la classe d'un élément en fonction de la classe majoritaire de ses k plus proches voisins.
- **Commentaires** : Il s'agit d'un exemple d'algorithme d'apprentissage.

Programme de terminale NSI (projet)

Les projets réalisés par les élèves (...) peuvent porter sur des problématiques issues d'autres disciplines ... Il peut s'agir (...) d'exploitation de **modules liés à l'intelligence artificielle** et en particulier à l'apprentissage automatique ...

2. Méthode des k plus proches voisins

a) Position du problème

On dispose d'une table de données dite « **base de données d'apprentissage** » constituée de N lignes représentant chacune un « individu ». Pour chaque individu, on dispose de *n* caractéristiques (les « **entrées** » et d'une donnée représentant la classe (ou catégorie) à laquelle appartient l'objet (la « **sortie** »).

Chaque ligne est donc constituée de $n+1$ données.

A partir de ces données, il s'agit de construire un **modèle prédictif** prenant en entrée *n* valeurs correspondant aux caractéristiques d'un « individu » et donnant en sortie la **classe** à laquelle appartient l'objet étudié.

b) Principe du modèle des k plus proches voisins

Pour estimer la sortie associée à n entrées $(x_1, \ldots, x_n)$, la **méthode des k plus proches voisins** consiste à déterminer les k lignes de la table d'apprentissage dont les *n* entrées sont les plus proches des valeurs $(x_1, \ldots, x_n)$.

On calcule ensuite la catégorie/classe la plus représentée parmi les k sorties associées aux k résultats précédents. Cette catégorie constitue le résultat de l'algorithme.

c) Quelques repères en Intelligence artificielle

Nous avons dit que l'algorithme des k plus proches voisins est un algorithme utilisé dans le domaine de l'IA comme méthode d'apprentissage supervisée. Que recouvre ce terme d'apprentissage supervisée ? Voici quelques repères :

L'intelligence artificielle (IA) est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence » (Définition de l'Encyclopédie Larousse)

l'apprentissage automatique (« Machine Learning ») est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches statistiques pour donner aux ordinateurs la capacité d'« apprendre » **à partir de données**, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. (Définition de Wikipédia)

Il y a différents types d'apprentissage automatique parmi lesquels :

- l'apprentissage supervisé,
- l'apprentissage non supervisé,
- l'apprentissage par renforcement,
- l'apprentissage par transfert,
- Le « deep learning »,
- ...

L'apprentissage supervisé est une tâche d'apprentissage automatique consistant à apprendre une fonction de prédiction à partir de données sur lesquelles on possède des mesures et pour lesquelles on connaît la classe (si les classes sont discrètes on parle de classification, si les classes sont continues, on parle de régression).

L'apprentissage non supervisé est une tâche d'apprentissage automatique pour laquelle on ne possède pas de données déjà classées pouvant servir de base à la prédiction.

3. Activité élève

1) Introduction

D'après wikipedia, « *l'**apprentissage automatique** (en anglais machine learning, littéralement « l'apprentissage machine ») est un champ d'étude de **l'intelligence artificielle** qui se fonde sur des approches statistiques pour donner aux ordinateurs la **capacité d' « apprendre »** à partir de **données**, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune*. »

Le développement d'internet et des **big data** permet la récupération et l'exploitation de nombreux jeux de données nécessaires à la conception d'un tel système.

Question 1

Avez-vous déjà entendu parlé d'intelligence artificielle, d'apprentissage automatique et de big-data ? Dans quel contexte ?

Commentaire

→ Débat dans la classe...

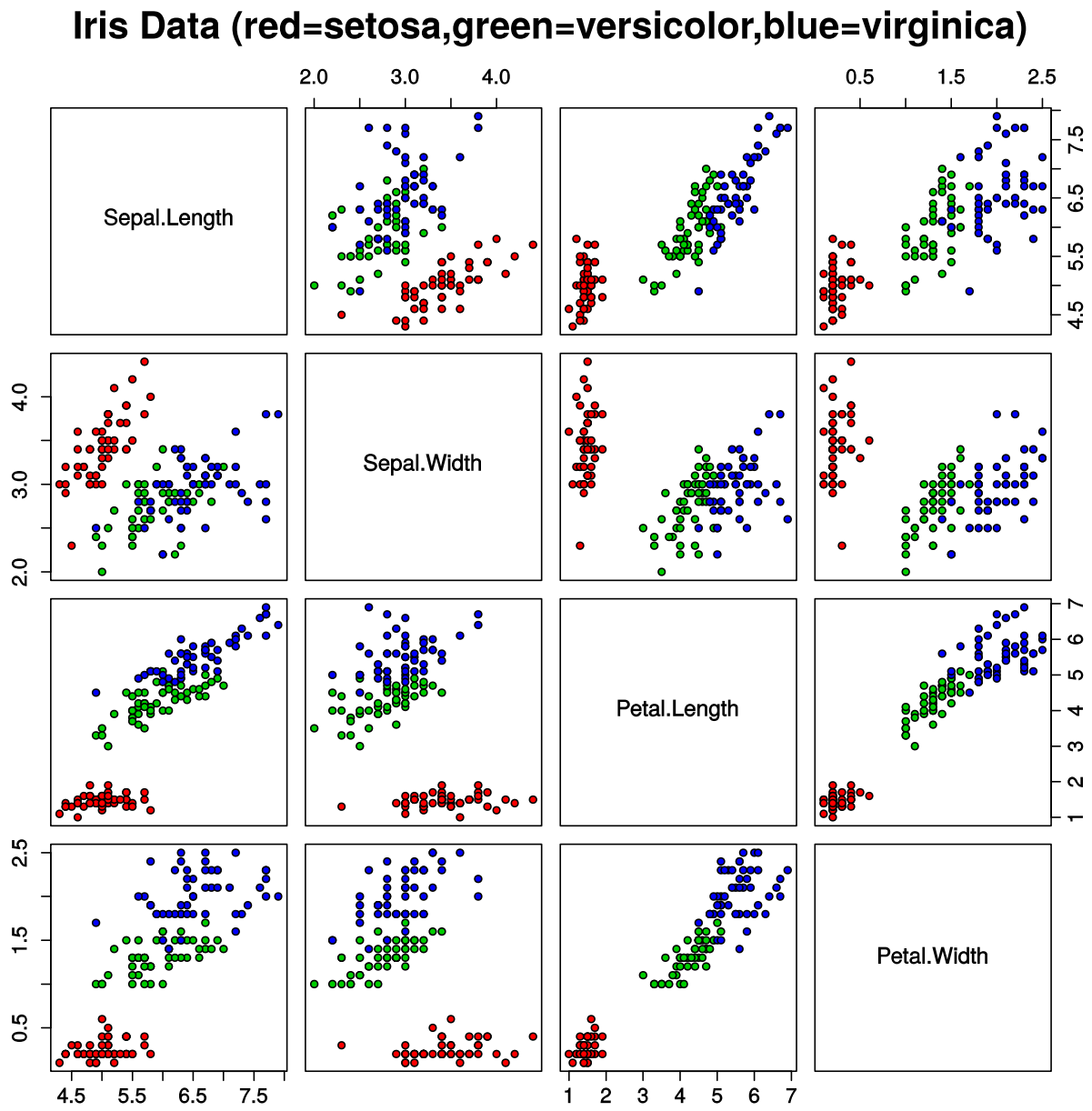
2) Position du problème

L'« **algorithme des k plus proches voisins** » que nous allons étudier maintenant (en anglais « k-NN », diminutif de « k Nearest Neighbors ») est un algorithme utilisé dans le domaine de **l'intelligence artificielle** comme méthode **d'apprentissage automatique**.

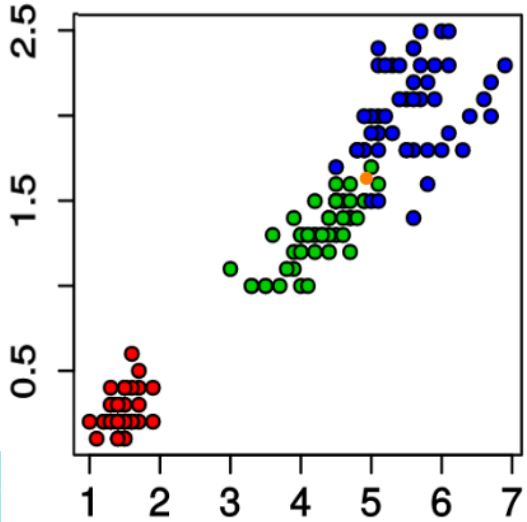
Pour comprendre l'intérêt de cet algorithme, nous allons étudier un jeu de données souvent utilisé dans le monde du machine learning, le jeu de données « Iris » (https://fr.wikipedia.org/wiki/Iris_de_Fisher)

Le jeu de données comprend 50 échantillons de chacune des trois espèces d'iris (Iris setosa, Iris virginica et Iris versicolor). Quatre caractéristiques ont été mesurées à partir de chaque échantillon : la longueur et la largeur des sépales et des pétales, en centimètres.

Le graphique ci-dessous visualise différents nuages de points en prenant en abscisses et ordonnées selon les cas, longueur et largeur des sépales et pétales. Les différentes espèces sont repérées par des couleurs différentes. On remarque que sur les différents graphiques, les points sont regroupés par espèces d'iris en « clusters ».



Question 2



Supposons à présent que l'on observe dans la nature, lors d'une promenade, une iris et que l'on mesure la longueur et la largeur des pétales : Comment déterminer à partir des données précédente à quelle espèce elle appartient ?

- Que dire d'une iris ayant telle que l=2 et L=0,5 ?
- Que dire d'une iris représentée par le point orange ci-contre ? Quelle méthode peut-on proposer pour déterminer la catégorie à laquelle appartient l'Iris ?

Commentaires

Le but de cette question sous la forme de débat est de faire émerger la méthode des k plus proches voisins.

3) Algorithme des k proches voisins

Une manière de répondre à la question est d'utiliser l'algorithme des k plus proches voisins :

On détermine les k plus proches voisins (k est fixé au début) : L'espèce la plus présente parmi ces k voisins est celle attribuée à notre Iris.

Nous allons mettre en œuvre cette méthode sur un jeu de données issu de données météorologiques.

Rendez-vous sur à l'adresse : https://www.infoclimat.fr/stations-meteo/analyses-mensuelles.php?annee=2019&mois=5&tri=libelle&s=asc

Ce site permet de collecter pour un mois donnée toutes les données (températures, précipitations, ensoleillement, vent) de toutes les stations météo en France. Le fichier « **meteo_mai_2019_France.csv** » mis à votre disposition contient trois colonnes :

- colonne 1 : le numéro du département où se situe la station,
- colonne 2 : la température moyenne au mois de mai 2019 enregistrée dans cette station,
- colonne 3 : la précipitation moyenne au mois de mai 2019 enregistrée dans cette station.

Supposons que vous ayez votre propre station météo quelque part en France et que disposiez pour le mois de mai d'un couple (température moyenne, précipitation moyenne). Nous allons essayer, à l'aide de la méthode décrite ci-dessus, de déterminer où se situe approximativement la station.

Dans la suite on note (t, p) ce couple.

Commentaires :

Il aurait été plus judicieux d'avoir des « moyennes de moyennes de températures » sur plusieurs années.

A noter ici le très grand nombre de catégories/classes (correspondant aux départements). Question : est-ce que la méthode k-NN est adaptée à un grand nombre de classe ? La quantité de données est-elle suffisante ?

Enfin, on ne s'attend pas nécessairement à retrouver exactement la bonne catégorie mais une catégorie au moins voisine.

Question 3

A l'aide du module CSV, écrire un programme qui range dans un tableau « data », les températures, les précipitations et le numéro du département sous la forme [[12.5, 80.6, '1'], [12.9, 92.6, '33'], ...].

Attention, certaines données (T ou P) sont manquantes pour certaines stations.

```
data=[]
# On crée un objet "reader"
with open('C:/temp/meteo_mai_2019_France.csv', 'r') as f:
  reader = csv.reader(f)
  for row in reader:
    if (row[1]!="") and (row[2]!=""):
            temp=[row[1],row[2],row[0]]
            T.append(temp)
```

Question 4 - Une première fonction distance

Chaque station peut être représentée par un point du plan.

a) Rappeler la formule permettant de calculer la distance entre deux points dont on connaît les coordonnées.

b) Ecrire une fonction python nommée « **distance2** » qui, étant donnés deux points du plan définis par leur coordonnées dans un repère orthonormé, renvoie le carré de la distance entre ces deux points.

Solution

```
def distance2(p,q):
  # entree: 2 listes de longueur 2 , retourne le carre de la distance euclidienne
  return (p[0]-q[0])**2+(p[1]-q[1])**2
```

Question 5 – Une liste des distances

Soit « data » la liste définie au-dessus et « individu » un tuple (t, p) de température et de précipitation.

Ecrire une fonction « **Liste_Distances(data,individu)** » qui renvoie sous la forme d'une liste, les distances (au carré) du tuple « individu » à chaque point formé à partir des données de « data ».

Solution

```
def liste_Distances(data, individu):
  l=[]
  n=len(data)
  for i in range(n):
    l.append([distance2(individu,data[i][0:2]),data[i][2]])
  return(l)
```

Question 6 – Une fonction des « k plus proches voisins »

Écrire une fonction « **lesKplusProchesVoisins** » qui prend en paramètre « data » et « individu » définis plus haut et une valeur de k et qui renvoie la liste des indices dans « data » des k plus proches voisins.

Indication :

- Commencer par écrire un algorithme (sur une feuille!)
- Éventuellement, commencer par écrire un algorithme qui renvoie l'indice du plus proche voisin (k=1)

Commentaires :

- Première solution : algorithme sans ordonner les distances
- Deuxième solution : algorithme (plus intelligent ?) consistant à commencer par ordonner le liste des distances.

Question : le deuxième algorithme est-il vraiment plus performant que le premier ?

La question du coût/complexité de l'algorithme est une question importante car les données manipulées en IA peuvent être de l'ordre du million.

Interroger les élèves sur le « coût » de ces algorithmes. Après analyse, le premier algorithme a un coût linéaire de l'ordre de $k \times n$

En classe de première les seuls algorithmes étudiés (sélection et insertion) ont un coût quadratique. Ces algorithmes ont un coût quadratique. Mais la fonction tri envisagée ici ne trie que les k premiers termes. Une analyse rapide permet de voir que l'algorithme avec tri a a également coût linéaire de l'ordre de $k \times n$.

Solution :

```
####################
```

```
# Solution sans tri #
#####################
def lesKplusProchesVoisins_sans_tri (data, individu,k):
 listeDistances = liste_Distances(data, individu)
 Kppv = []
 liste_indices_utilises=[]
 for i in range (k):
  p = float ("inf")
  for j in range (len (data)):
   if listeDistances [j][0] != 0 and listeDistances [j][0] < p and j not in
liste_indices_utilises:
    p = listeDistances [j][0]
    indice = j
  Kppv. append (listeDistances [indice][1])
  liste_indices_utilises.append(indice)
 return (Kppv)

k=5
individu=(12.0,50.0)
listeKppv = lesKplusProchesVoisins_sans_tri(data,individu,k)


#####################
# Solution avec tri #
#####################
# Fonctions auxiliaires
def estIndice(T,i):
  return(0<=i and i<len(T))

def echange(T,i,j):
  assert(estIndice(T,i) and estIndice(T,j))
  aux = T[i]
  T[i] = T[j]
  T[j] = aux

def triSelectionModif(data, individu,k): # on ne trie que les k premiers termes
    T=liste_Distances(data, individu)
    for i in range(k-1):
        iMin=i
        for j in range(i+1,len(T)):
            if T[j][0]<T[iMin][0]:
                iMin = j
        if iMin != i :
            echange(T,i,iMin)
    return(T)

def lesKplusProchesVoisins_avec_tri (data, individu,k):
 assert(k<=len(data))
 L=triSelectionModif(data,individu,k)
 return [L[i][1] for i in range(k)]

k=5
individu=(12.0,50.0)
listeKppv = lesKplusProchesVoisins_avec_tri(data,individu,k)
print(listeKppv)
```

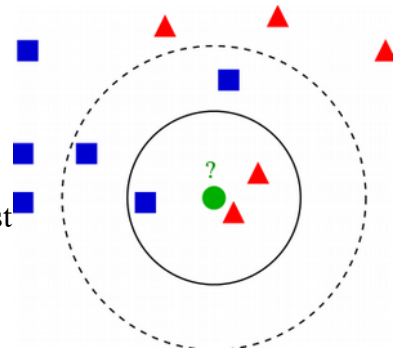
Question 7 – Prédire la classe

Il vous reste à écrire une fonction « prediction » qui prend en paramètre « data » et « individu » définis plus haut et une valeur de k et qui renvoie la catégorie prédite (ici le département).

```
def  prediction_avec_tri(data,individu, k):
```

```
    etiquettesKvoisins=lesKplusProchesVoisins_avec_tri(data,individu,k)
    lstEtiquettes=[etiquettesKvoisins[0]]#liste sans doublon des etiquettes des k voisins
    lstOccurrences=[1]#liste des occurences des etiquettes
    occMax=1
    indMax=0#pour recuperer l'etiquette correspondant à occMax
    #construction de la liste sans doublons et recherche de l'etiquette la plus presente
    for i in range (1,k):
        n=len(lstEtiquettes)
        trouve=False# l'etiquette testee est elle deja dans la liste?
        for j in range(n):
            if etiquettesKvoisins[i] == lstEtiquettes[j]:
                lstOccurrences[j]+=1
                if lstOccurrences[j]>occMax:
                    occMax=lstOccurrences[j]
                    indMax=j
                trouve=True
        if not trouve:
            lstEtiquettes.append(etiquettesKvoisins[i])
            lstOccurrences.append(1)
    return lstEtiquettes[indMax]


def prediction_sans_tri(data,individu, k):
    etiquettesKvoisins=lesKplusProchesVoisins_sans_tri(data,individu,k)
    lstEtiquettes=[etiquettesKvoisins[0]]#liste sans doublon des etiquettes des k voisins
    lstOccurrences=[1]#liste des occurences des etiquettes
    occMax=1
    indMax=0#pour recuperer l'etiquette correspondant à occMax
    #construction de la liste sans doublons et recherche de l'etiquette la plus presente
    for i in range (1,k):
        n=len(lstEtiquettes)
        trouve=False# l'etiquette testee est elle deja dans la liste?
        for j in range(n):
            if etiquettesKvoisins[i] == lstEtiquettes[j]:
                lstOccurrences[j]+=1
                if lstOccurrences[j]>occMax:
                    occMax=lstOccurrences[j]
                    indMax=j
                trouve=True
        if not trouve:
            lstEtiquettes.append(etiquettesKvoisins[i])
            lstOccurrences.append(1)
    return lstEtiquettes[indMax]
```

Question 8 – Choix de k



a) En étudiant l'exemple ci-contre, expliquer pourquoi le choix de k détermine le résultat de l'algorithme.

Ce paramètre k ne peut être appris automatiquement par l'algorithme à partir des données d'apprentissage. Une façon d'optimiser le choix de k est d'utiliser des jeux de données test.

b) Voici quelques données test (qui ne figurent pas dans les données d'apprentissage!). Tester votre modèle et ajuster la valeur de k pour optimiser les prédictions.

Station	**Département**	**Temp. Moyenne**	**Préc. Moyenne**
Nice - Côte d'Azur (06)	6	16,2	20
Arvert (1) (17)	17	14,6	22,8
Senlis (60)	60	12,6	93,3
Quimper-Pluguffan (29)	29	12,9	51,1
Biarritz-Anglet (64)	64	14,1	178,2
Saint-Laurent-du-Pont (38)	38	12,3	183,2
Figari (2A)	2A	15,8	34,6

Commentaires :

Avec k=30 on obtient les résultats suivants :

Station	Département	Prédiction
Nice - Côte d'Azur (06)	6	83
Arvert (1) (17)	17	17
Senlis (60)	60	78
Quimper-Pluguffan (29)	29	56
Biarritz-Anglet (64)	64	38
Saint-Laurent-du-Pont (38)	38	38
Figari (2A)	2A	26

4. Utilisation de la bibliothèque Python Scikit Learn

Python dispose de quelques bibliothèques spécialisées en IA et notamment la bibliothèque « **Scikit Learn** » qui propose un grand nombre d'algorithmes liés au machine learning. Il s'agit d'une des bibliothèques les plus utilisées en machine learning.

Parmi tous ces algorithmes, « Scikit Learn » propose l'algorithme des k plus proches voisins.

Installation de Scikit Learn en ligne de commande :

```
pip install -U scikit-learn
```

ou bien, dans la cas d'une installation « anaconda » :

```
conda install scikit-learn
```

Question 9 – Utiliser la bibliothèque Scikit Learn

a) Etudier et tester le programme suivant.
b) Vérifier que vous retrouver les mêmes résultats qu'avec votre programme.

```
# on importe le module CSV.
import csv
# on importe le module pyplot de la bibliothèque matplotlib
import matplotlib.pyplot as plt
# on importe la classe KNeighborsClassifier du module neighbors (bibliothèque sklearn)
from sklearn.neighbors import KNeighborsClassifier
```

```
departement=[]
d=[]
# On crée un objet "reader"
with open('C:/data/Dropbox/data1/Numérique et informatique/formation NSI/Bloc 2 -
Algo\projet/meteo_mai_2019_France.csv', 'r') as f:
  reader = csv.reader(f)
  for row in reader:
    if (row[1]!="") and (row[2]!=""):
      temp=[row[1],row[2]]
      d.append(temp)
      departement.append(row[0])
# Description de meteo_mai_2019_France.csv :
# colonne 0 : Num département
# colonne 1 : Moy Temp
# colonne 2 : Cumul Préc

# Choix de k
k=30
# On crée une instance de l'objet KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=k)
# On applique la méthode "fit" pour remplir le modèle
model.fit(d,departement)

# Test avec Nice - Côte d'Azur (06)
moy_temp=16.2
cumul_prec=20.0
vent=74.9
# On lance le modèle afin d'effectuer une prédiction
prediction= model.predict([[moy_temp,cumul_prec]])
# On affiche le résultat
print ("Nice - Côte d'Azur (06) - valeur prédite : ", prediction[0])
```

5 Approfondissement - Evaluation sur un jeu de données

On peut évaluer les différents programmes d'implémentation de la méthode sur un jeu de données fictif en scindant celui-ci en deux jeux:

Le premier servira de **jeu de données d'apprentissage** pour l'algorithme et le second **jeu de données de test** servira à tester l'algorithme avec, à la clef, l'affichage d'un pourcentage de réussite.

Dans l'exemple qui suit on a généré à l'aide du module random de python un jeu de points du plan que l'on a découpé à l'aide de fonctions usuelles en 6 catégories.

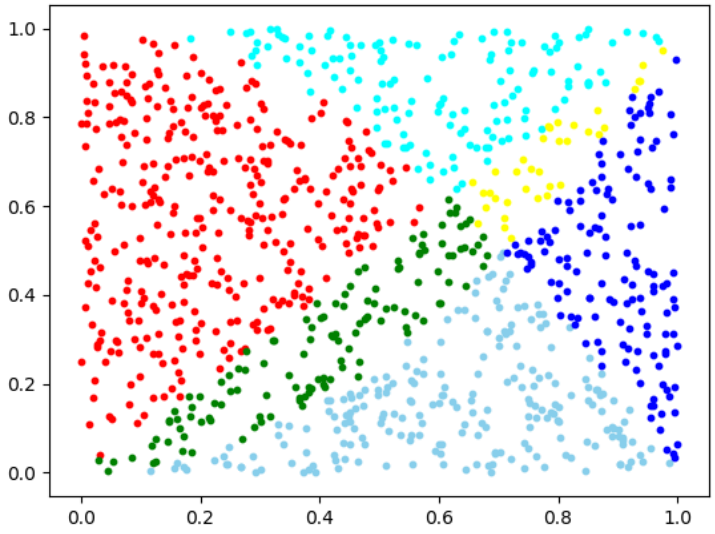
Pour scinder le jeu de donnés en deux jeux (apprentissage et test), on applique un coefficient (par exemple 0,8 pour prendre 80% des données pour l'apprentissage, les 20% restant représentant les données de test.

```
import matplotlib.pyplot as plt
from random import random
import Projet_Bloc2_Carcone_Delineau_Algorithme_KNN

n=1000

size=10

xa,xb,xc,xd,xe,xf,ya,yb,yc,yd,ye,yf=[],[],[],[],[],[],[],[],[],[],[],[]
T=[]
for i in range(n):
        x=random()
        y=random()
```

```
        if y<1-x**2:
            if y>0.5*(x+y):
                xa+=[x]
                ya+=[y]
                T.append([x,y,'a'])
            elif y>x**2:
                xb+=[x]
                yb+=[y]
                T.append([x,y,'b'])
            else:
                xc+=[x]
                yc+=[y]
                T.append([x,y,'c'])
        else:
            if y>0.5*(x+y):
                xd+=[x]
                yd+=[y]
                T.append([x,y,'d'])
            elif y>x**2:
                xe+=[x]
                ye+=[y]
                T.append([x,y,'d'])
            else:

                xf+=[x]
                yf+=[y]
                T.append([x,y,'e'])

plt.scatter(xa,ya,s=size,c='red')
plt.scatter(xb,yb,s=size,c='g')
plt.scatter(xc,yc,s=size,c='skyblue')
plt.scatter(xd,yd,s=size,c='cyan')
plt.scatter(xe,ye,s=size,c='yellow')
plt.scatter(xf,yf,s=size,c='b')
plt.show()


coeff=0.8
data=T
k=10

def pourcentageReussite_avec_tri(data,coeff,k):
    n=len(data)
    tabRef=data[:int(coeff*n)]
    tabTest=data[int(coeff*n):]
    nbTestOk=0
    for individu in tabTest:
        etiq= prediction_avec_tri(data,individu,k)
        if etiq==individu[2]:
            nbTestOk+=1
    return nbTestOk*100/len(tabTest)

def pourcentageReussite_sans_tri(data,coeff,k):
    n=len(data)
    tabRef=data[:int(coeff*n)]
    tabTest=data[int(coeff*n):]
    nbTestOk=0
    for individu in tabTest:
        etiq= prediction_sans_tri(data,individu,k)
        if etiq==individu[2]:
            nbTestOk+=1
    return nbTestOk*100/len(tabTest)
```

```
print("pourcentage de reussite sans tri:",pourcentageReussite_sans_tri(data,coeff,k))
print("pourcentage de reussite avec tri:",pourcentageReussite_avec_tri(data,coeff,k))
```

Dans le cas $n=1000$, voici un exemple de nuage de points générés aléatoirement et faisant apparaître les 6 classes :



En faisant prendre à *n* de grandes valeurs ont peut également comparer les temps d'exécution de chacun des deux algorithmes. Avec n=1000000, k=30, le temps d'exécution sur un ordinateur portable de moyenne gamme est de l'ordre de la dizaine de secondes pour les deux algorithmes.

Enfin, il est à noter que la bibliothèque « scikit learn » de Python met à disposition de l'utilisateur une méthode permettant de tester les performances de prédiction de la méthode k-nn dans lequel on passe un jeu de données de test. Cette méthode renvoie le pourcentage de prédiction véridique trouvée.