Files d'attente, algorithme PPV
Devoir à faire à la maison
Année 2019-2020

Ce devoir est à réaliser à la maison pendant la semaine du 9 au 13 mars 2020. Il est à rendre pour le vendredi 20 mars (voir espace de rendu sur Moodle : Devoir Maison)

1 Utilisation d'une file pour simuler la gestion d'une file d'attente

Soit une file d'attente à un guichet unique (type file d'attente d'un bureau de poste). On considère les hypothèses suivantes :

- Le guichet est ouvert 8h00 consécutives ;
- Les clients arrivent et font la queue. La probabilité d'arrivée d'un nouveau client dans l'intervalle [t...t+1] est p. Le temps sera évalué en secondes (1 seconde = 1 unité de temps dans la boucle de simulation) ;
- Le temps que prend le service d'un client suit une loi uniforme sur [30...300] (en secondes)

Rappel : vous pourrez utiliser les fonctions random() et uniform() des librairies suivantes :

```
from random import uniform
from random import random
```

Ecrivez la classe `SimulFileAttente` qui permet de modéliser les clients et de simuler la file d'attente des clients :

1. Modélisez la classe `SimulFileAttente` qui contient au sein de son constructeur les différentes constantes temporelles nécessaires à la gestion de la file d'attente : `currentTime` est la variable qui mesure le temps courant dans la boucle de simulation (incrémenté de 1 à chaque itération) ; `tMax` est la durée maximale d'ouverture du guichet (en secondes), `serviceMin` et `serviceMax` sont les temps temps de service minimal et maximal d'un client, ces temps étant exprimés en secondes.

2. Modélisez au sein de la classe `SimulFileAttente` la classe `Client` ; le constructeur a pour paramètre le temps `t` courant et calcule l'heure de départ `leaveTime` du client à partir de son heure d'arrivée et des constantes d'attente minimale et maximale (`waitMin` et `waitMax`) obtenues par une loi uniforme.

3. Ecrivez la méthode `simul` de la classe `SimulFileAttente` qui prend en paramètre la probabilité `p` d'arrivée d'un client et retourne le rapport entre nombre de client servis et le nombre total de clients pendant une journée de 8 heures.
   Indication : vous entrerez un nouveau client dans la file d'attente si le nombre aléatoire

fourni par la fonction `random` est plus petit que `p`. Vous pourrez tester cette méthode en prenant des valeurs spécifiques de probabilité. En particulier vous pourrez vérifier qu'au delà d'une certaine probabilité certains clients ne seront pas servis (grande affluence).

4. Modifiez le programme précédent en écrivant une méthode `simul_bis` qui tient compte du fait que les clients ont une patience limitée. En effet, si les clients attendent trop longtemps, ils partent sans être servis. Vous pourrez considérer que leur patience est une loi uniforme sur [120...1800]. Dans cette question, vous évaluerez la patience des clients au moment où ils arrivent en tête de la file. La fonction `simul_bis` retournera le rapport du nombre de clients non servis sur le nombre total de clients.

5. Modifiez le programme précédent en écrivant une méthode `simul_ter` qui, à chaque itération, retire de la file les clients qui ont dépassé leur quota de patience.

6. Ecrivez la méthode `main` qui teste pour plusieurs probabilités (à partir de 0.00025 jusqu'à 0.025, par pas de `delta=0.00025`) la valeur du rapport précédent (nombre de clients non servis sur le nombre total de clients), et tracez la courbe correspondante.

2 Algorithme de recherche des plus proches voisins

On s'intéresse à un algorithme de classification par une méthode dites des Plus Proches Voisins qui s'applique sur des données d'apprentissage (ou d'entraînement). Pour un ensemble de `n` données d'entraînement, on est capable de partitionner l'espace selon l'appartenance des données aux classes. Par exemple, sur la figure 1, les points représentés en rouge (étoiles) appartiennent à la classe A et les points en vert (triangles) appartiennent à la classe B. Un nouveau point peut être classé dans l'une des deux classes A ou B, en fonction de ses plus proches voisins (le nombre `k` constituant un paramètre de l'algorithme). Pour ce nouveau point qui n'appartient pas aux données d'entraînement, on détermine parmi les plus proches voisins quelle est la classe majoritaire. Par exemple, si $\texttt{k} = 3$, et que l'on trouve 3 plus proches voisins associés aux classes respectives (A,A,B), alors ce nouveau point sera classé A.

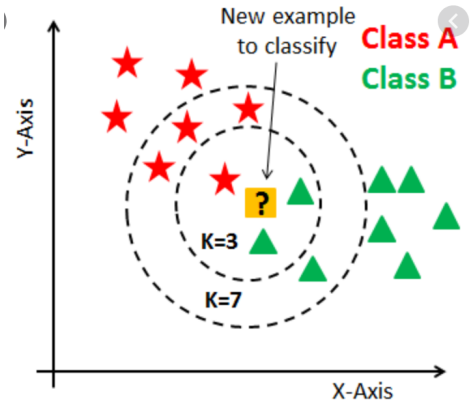

Figure 1: Exemple de classification par méthode k-PPV dans un plan

Dans le cadre de cet exercice, chaque donnée est caractérisée par un point en 2 dimensions, associé à une classe (identifiée par un numéro). Nous proposons de définir les différents niveaux de données par des types abstraits (codés au moyen de classes en Python). Nous représentons

ainsi les points (type abstrait Point), les données (type abstrait Data) et l'algorithme de classification (type abstrait `KPPV`) permettant de déterminer les plus proches voisins et de prédire la classe d'une nouvelle donnée de test.

1. Complétez la classe `Point` qui comporte les attributs x, y , ainsi que les méthodes `__init__`, `__str__`, et la méthode `distance(self, p)` retournant la distance entre deux points. Vous prendrez la distance euclidienne.

2. Complétez la classe `Data` qui comporte les attributs `point` et `classe`, ainsi que les méthodes `__init__`, `get_point()` et `get_classe()`.

3. Complétez la classe `KPPV`, qui comporte les méthodes `__init__` (constructeur) et `add_data` qui permet d'ajouter une donnée de type Data à l'ensemble d'entraînement `trainingSet`.

4. Écrivez la méthode `get_voisins` qui prend en entrée un point `p_test` et un entier `k` et retourne la liste des `k` données de type Data les plus proches du point `p_test`.

5. Écrivez la méthode `prediction` qui prend en entrée un point `p_test` et un entier `k` et retourne la classe prédite par le classifieur `KPPV`.

Attention – Vous tiendrez compte dans cet exercice des types de données imposés.
Par exemple, si l'on considère d'ensemble d'entraînement suivant :

```
dataset = [Data(Point(2.7810836,2.550537003),0),
Data(Point(1.465489372,2.362125076),0),
Data(Point(3.396561688,4.400293529),0),
Data(Point(1.38807019,1.850220317),0),
Data(Point(3.06407232,3.005305973),0),
Data(Point(7.627531214,2.759262235),1),
Data(Point(5.332441248,2.088626775),1),
Data(Point(6.922596716,1.77106367),1),
Data(Point(8.675418651,-0.242068655),1),
Data(Point(7.673756466,3.508563011), 1)]
```

L'appel de la méthode `get_voisins` pour le point test (5.0, 2.0) et $k = 3$ renverra les données suivantes (liste de couples (point, classe), et la classe prédite pour ce point test sera la classe 1 (classe représentée majoritairement).

```
[5.332441248,2.088626775] 1
[6.922596716,1.77106367] 1
[3.06407232,3.005305973] 0
```