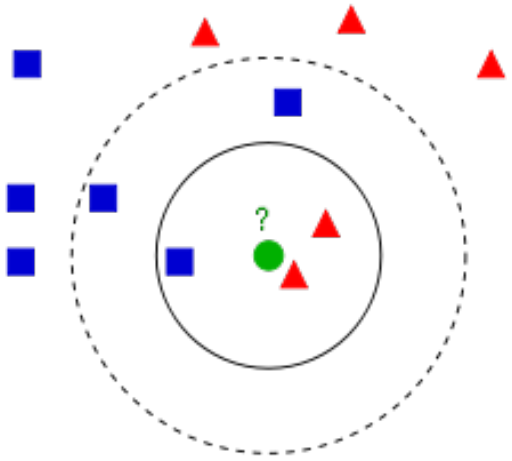
Chapitre 3

Algorithme des k plus proches voisins



3.1 Présentation de l'algorithme

L'algorithme des k **plus proches voisins**, aussi appelé **algorithme K-NN** (pour *K Nearest Neighbors* en anglais) est un **algorithme d'apprentissage supervisé**. Son principe de fonctionnement peut se résumer en une phrase :

« Dis moi qui sont tes voisins, je te dirai qui tu es »

À partir d'un jeu de données, l'algorithme est capable d'effectuer une prédiction (une classification ou une régression ici). Il s'agit donc d'un algorithme de **machine learning**, qui *apprend* à partir des données qui lui sont fournies.

Concrètement, voici quelques exemples d'application de l'algorithme K-NN :

1. Détection des SPAMS : l'algorithme analyse la composition d'un mail. Notamment, le contenu de ce dernier, ainsi que le nombre d'occurrences des mots le constituant, etc ... À la suite de cette analyse, l'algorithme décidera si un mail est un SPAM ou non.
2. Diagnostic médical : en se basant sur les données médicales d'un patient, l'algorithme peut diagnostiquer si le sujet est atteint d'une maladie donnée. Parfois, un tel algorithme peut alerter d'un incident grave de santé avant qu'il n'arrive.
3. Détection de fraudes
4. Proposition d'articles sur un site Web marchand en fonction des articles déjà vus / achetés
5. Proposition de vidéo sur Youtube en fonction des vidéos déjà vues

Comment l'algorithme effectue-t-il une prédiction ?

À partir du jeu de données dont il dispose, l'algorithme va calculer les K valeurs d'entrées **les plus proches** d'une observation x donnée. Un résultat y prédit pour cette observation est alors calculé à partir des valeurs de sortie de ces K valeurs d'entrées. Ce résultat peut être :

- la valeur dominante des valeurs de sortie : on parle alors de **classification**, car la valeur y associée à notre observation est égale à une des valeurs de sortie de nos exemples du jeu de données
- une moyenne des valeurs de sortie : on effectue alors une **régression**, c'est à dire qu'on associe un nombre réel y à notre observation

▷ ***Je ne comprends rien... Peut-on prendre un exemple simple ?***

Mais bien sûr ! Découvrons sans plus attendre 2 exemples : un exemple de classification et un exemple de régression.

Un exemple de classification : réussite à un examen

Suite à un examen, on interroge 20 élèves à propos du temps qu'il ont passé à étudier et du temps qu'ils ont passé à se reposer. On leur demande également s'il ont réussi l'examen, puis ces données sont regroupées dans le tableau suivant :

Valeurs d'entrées		*Valeur de sortie*
Heures de travail	Heures de repos	Réussite à l'examen
2,5	4	✓
1	8	×
5	3	×
…	…	…
4	1	✓

À votre tour de passer l'examen : vous avez travaillé 3h et vous êtes reposé 5h. Aurez-vous votre examen ?

Valeurs d'entrées		*Valeur de sortie*
Heures de travail	Heures de repos	Réussite à l'examen
3	5	?

Bien entendu, impossible de deviner sans erreur votre résultat, mais il est néanmoins possible de réaliser une prédiction, à partir des données recueillies sur les élèves ayant déjà passé leur examen.
Ici, le jeu de données est donc l'ensemble des données recueillies auprès des élèves, et l'observation... c'est vous !

▷ ***Ça n'explique toujours pas comment fait l'algorithme pour prévoir le résultat...***

En effet... Pour le comprendre, il est nécessaire de *voir* notre jeu de données. Rien de tel qu'un graphique.

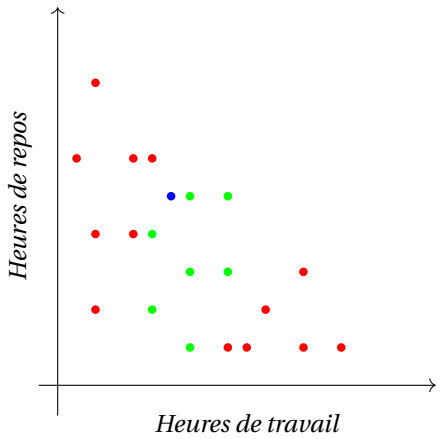
Sur la graphique ci-contre, on a représenté la réussite par un point vert et l'échec par un point rouge. L'abscisse d'un point correspond au nombre d'heures travaillées, et l'ordonnée le nombre d'heures de repos.

Quant au point bleu, c'est vous, l'observation.

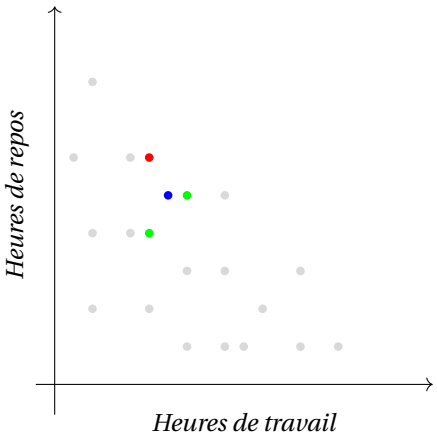
Ici, on a un problème de classification, car on cherche à savoir si votre examen sera réussi ou non, autrement dit si le point bleu doit plutôt être rouge ou vert.

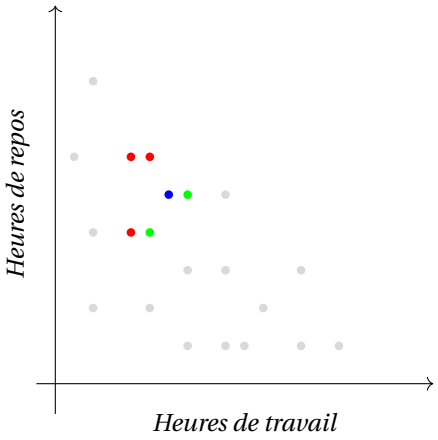
Pour savoir cela, il suffit de regarder les « plus proches voisins » du point bleu. Quels sont les points correspondant ?

Tout dépend du nombre de voisins choisis !



En regardant les **trois** plus proches voisins, vous réussissez votre examen, car parmi eux, deux sont verts et un seul est rouge. La valeur retenue pour votre cas est la valeur dominante des plus proches voisins, ici la réussite.





En regardant les **cinq** plus proches voisins, vous échouez à votre examen, car parmi eux, deux sont verts et mais trois sont maintenant rouges. La valeur retenue pour votre cas est la valeur dominante des plus proches voisins, ici l'échec.

☞ *Comprenez-vous à présent l'image sur la première page de ce document ?*

▷ ***Alors combien de voisins doit-on choisir ? 3 ou 5 ? ou plus ?***

Et bien c'est à vous de choisir ! En général, on choisit le nombre de voisins qui permet de représenter au mieux le jeu de données. Préférez cependant un nombre impair de voisins : dans le cas d'une classification binaire comme dans cet exemple, il ne peut pas y avoir d'indécision !

▷ ***Comment peut-on savoir quels sont les points les plus « proches » ?***

Il faut pour cela introduire une **distance**. Dans notre problème, il s'agit simplement de la distance entre deux points du plan (on parle de *distance euclidienne*), dont la formule a été vue en classe de Seconde en Mathématiques :

$$AB = \sqrt{(x_B - x_A)^2 + \left(y_B - y_A\right)^2} \qquad A(x_a; y_A) \;\; B(x_B; y_B)$$

Un exemple de régression : note à un examen

Dans le problème précédent, la prédiction portait sur la réussite ou l'échec à l'examen : on parlait alors de classification (classification *binaire* dans ce cas précis). Supposons maintenant que les élèves aient récupéré leurs notes :

Valeurs d'entrées		*Valeur de sortie*
Heures de travail	Heures de repos	Note à l'examen (/20)
2,5	4	16,4
1	8	7,3
5	3	9,5
...	...	...
4	1	11,9

Vous qui avez travaillé 3h et vous être reposé 5h, quelle serait alors votre note à l'examen ?

Valeurs d'entrées		*Valeur de sortie*
Heures de travail	Heures de repos	Note à l'examen (/20)
3	5	?

Nous sommes à présent dans un problème de **régression** : il ne suffit plus de dire si vous aurez votre examen ou non, il faut à présent prévoir votre note, qui pourrait prendre un grand nombre de valeurs différentes.

Le calcul est différent mais le principe de l'algorithme reste le même : on va sélectionner les plus proches voisins (par exemple les 3 plus proches voisins) et on va effectuer la moyenne de leurs notes, ce qui donnera une prévision de votre résultat.

3.2 Codage de distances

Distance euclidienne

Comme nous l'avons vu, l'algorithme K-NN repose sur la calcul de distances. Dans le cas de deux valeurs d'entrées, on peut représenter graphiquement les exemples du jeu de données par des points du plan. La distance entre deux de ces points est simplement la distance euclidienne que vous connaissez.

Question 01 Écrire une fonction `distancePlan(P1, P2)` qui retourne la distance euclidienne entre deux points `P1` et `P2`, donnés sous forme de *tuple* ou de listes à deux éléments.

```
>>> distancePlan((1,1), (2,3))
2.23606797749979
```

Supposons qu'une troisième variable soit prise en compte dans le problème précédent, et que l'on demande par exemple aux étudiants, en plus du nombre d'heures travaillées et du nombre d'heures de repos, le nombre d'heures passées à faire du sport. Pour représenter nos données sur un graphique, il faudrait alors un troisième axe, et nos données seraient alors représentées non plus dans le plan mais **dans l'espace** (l'espace au sens Mathématique, il n'y a pas d'astronautes ici...).

Dans l'espace, tout point M est repéré par 3 coordonnées $(x; y; z)$. La distance entre deux points $A(x_A; y_A; z_A)$ et $B(x_B; y_B; z_B)$ est donnée par une formule analogue à la formule précédente :

$$AB = \sqrt{(x_B - x_A)^2 + \left(y_B - y_A\right)^2 + (z_B - z_A)^2}$$

Question 02 Écrire une fonction `distanceEspace(P1, P2)` qui retourne la distance entre deux points `P1` et `P2` **de l'espace**, donnés sous forme de *tuple* ou de listes à trois éléments.

```
>>> distanceEspace((1,1,1), (2,3,4))
3.7416573867739413
```

Si l'on rajoute une quatrième variable (comme le nombre d'heures passées à manger), il n'est plus possible de représenter nos données graphiquement (sauf si vous voyez en 4D). Mais mathématiquement, on peut toujours définir une distance, de façon analogue aux définitions précédentes. De façon plus générale, on peut toujours définir une distance euclidienne dans un espace à n dimensions.

▷ ***J'ai mal à la tête...***

Pas la peine d'en arriver là, ce n'est qu'une définition mathématique.
On définit la *distance euclidienne* entre deux points $A(x_1; x_2; \ldots; x_n)$ et $B(y_1; y_2; \ldots; y_n)$ par :

$$d = \sqrt{\sum_{k=1}^{n} (y_i - x_i)^2} = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \ldots + (y_n - x_n)^2}$$

Question 03 Écrire une fonction `distanceEuclienne(P1, P2)` qui retourne la distance euclidienne entre deux points `P1` et `P2` , donnés sous forme de *tuple* ou de listes à n éléments.

```
>>> distanceEuclienne((1,1,1,1), (2,3,4,5))
5.477225575051661
>>> distanceEuclienne((1,1,1,1,1), (2,3,4,5,6))
8.12403840463596
```

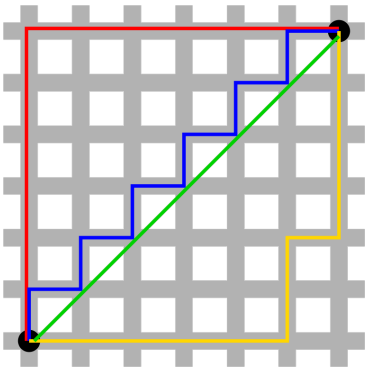
☞ *Le nombre n de coordonnées doit être déterminé automatiquement. Ainsi, on peut également utiliser cette fonction avec seulement 2 ou 3 coordonnées, et ainsi se passer de nos deux premières fonctions* `distancePlan` *et* `distanceEspace` .

Distance de Manhattan

La **distance de Manhattan**, appelée aussi *taxi-distance*, est la distance entre deux points parcourue par un taxi lorsqu'il se déplace dans une ville où les rues sont agencées selon un réseau ou quadrillage. Un *taxi-chemin* est le trajet fait par un taxi lorsqu'il se déplace d'un nœud du réseau à un autre en utilisant les déplacements horizontaux et verticaux du réseau.

Étant donnés deux points $A(x_A; y_A)$ et $B(x_B; y_B)$, la distance de Manhattan est donnée par :

$$d = |x_B - x_A| + |y_B - y_A|$$



Vert : distance euclidienne
Rouge,bleu,jaune : distance de Manhattan

Question 04 Écrire une fonction `distanceManhattan(P1,P2)` qui retourne la distance de Manhattan entre deux points `P1` et `P2` , donnés sous forme de tuple ou de listes à 2 éléments.

```
>>> distanceManhattan((1,1), (5,4))
7
```

3.3 Codage de l'algorithme

Définissons rigoureusement l'algorithme des k plus proches voisins :

- Données d'entrées :
  1. une observation X
  2. un jeu de données D
  3. une distance d
  4. un nombre entier k
- Pour une nouvelle observation X dont on veut prédire sa valeur de sortie y, faire :
  1. Calculer toutes les distances de cette observation X avec les autres exemples du jeu de données D
  2. Sélectionner les k plus proches exemples de X
  3. Calculer la valeur de sortie y prédite :
     (a) en déterminant la valeur dominante dans le cas d'une classification
     (b) en faisant la moyenne des valeurs de sortie dans le cas d'une régression
  4. Retourner la valeur de y ainsi déterminée

Pour ne pas rendre l'exercice trop difficile, on utilisera la distance euclidienne.

Question 05 Écrire une fonction `KNN(X,D,k)` où `X` est une observation (sous la forme de tuple ou de liste), `D` est l'ensemble des données (voir plus bas pour plus de détails) et `k` est le nombre plus proches voisins souhaité.

Exemple. Vous trouverez le jeu de données de l'exemple vu précédemment dans ce chapitre à l'adresse suivante :

`http://nsi.dellasantina.corsica/documents/algorithmique/jeu_donnees_knn.txt`

Le jeu d'exemple est une liste dont les éléments sont des listes de la forme `[(x,y), c]`, où `x` et `y` sont les coordonnées du point (heures de travail et heures de repos) et `c` est la **classe** associée au résultat à l'examen (0 pour échec, 1 pour réussite). Pour tester votre programme, vous pouvez utiliser l'ébauche de code ci-dessous :

```
# Algorithme des k plus proches voisins

def KNN(X,D,k):
    # À coder !

D = [[(5, 1), 0], [(4.5, 3), 1], ... , [(2, 4), 0]] # Jeu de données
X = (3,5) # Observation (3h de travail, 5h de repos)

print(KNN(X,D,3)) # Résultat avec 3 plus proches voisins
print(KNN(X,D,5)) # Résultat avec 5 plus proches voisins
```

3.4 Pour aller plus loin

Tracé des frontières