Les algorithmes d'apprentissage sont des algorithmes permettant de répondre à plusieurs types de problèmes :

— des problèmes dont l'énnoncé est intrinsèquement flou (par exemple trouver la meilleure traduction d'un texte dans une langue donnée) ;

— des problèmes dont on ne sait pas déterminer une solution exacte en un temps raisonnable et pour lesquels on peut se suffire d'une approximation ;

— des problèmes dont les données sont partielles.

1 Problèmes de classification

On considère deux ensembles C (classes) et E (éléments) et une fonction $c : E \longrightarrow C$ associant à chaque élément e une unique classe $c(e)$.

On parle de problème de classification lorsque l'on se donne comme objectif de prévoir la valeur de $c(e)$ pour tout $e \in E$.

Exemple (carte scolaire) : on cherche à prédire à quel établissement scolaire un élève d'une ville va être rattaché.

Exemple (l'algorithme de Youtube) : on cherche à prédire si un utilisateur de Youtube va aimer ou non une vidéo.

Sans plus d'information, il est impossible de résoudre de tels problèmes. C'est pourquoi on va également se doter de deux outils supplémentaires :

— Un ensemble $T \subset E$ d'éléments dont on connaît la classe (les données).

— Une distance $d : E \times E \longrightarrow \mathbb{R}_+$ permettant de quantifier une certaine notion de proximité entre les éléments de E.

Exemple (carte scolaire) : T représente une partie des élève dont on connaît déjà l'affectation. Un choix possible de d peut être la distance physique entre les domiciles des élèves.

Exemple (l'algorithme de Youtube) : T représente un jeu d'utilisateurs qui ont déjà visionné et apprécié ou non la vidéo. On peut définir d en se basant sur la proximité des profils (abonnements, vidéos aimés, etc.)

On voit à travers l'exemple des vidéos Youtube que le problème, bien qu'ayant une réponse bien définie (chaque personne va aimer ou non la vidéo), est extrêmement difficile à résoudre de manière exacte notamment car il comporte une composante humaine.

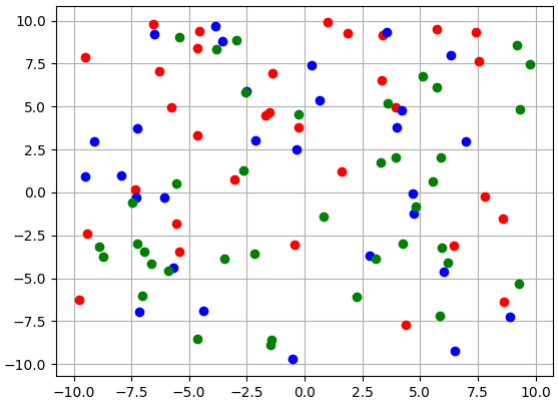
Le problème de la carte scolaire pourrait être résolu facilement si on connaissait précisément les critères d'affectation des élèves. On suppose cependant qu'on ne les connaît pas.

2 Algorithme des k plus proches voisins (k-NN)

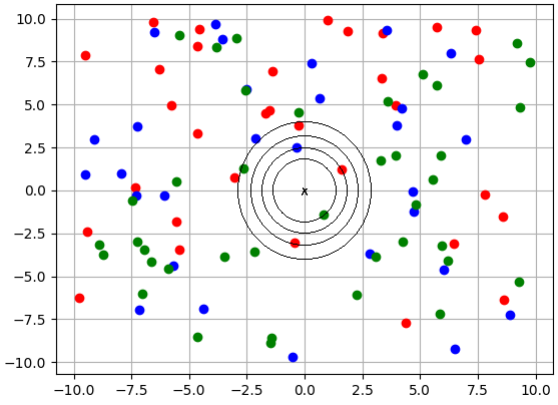
k-NN : principe Le principe de l'algorithme des k plus proches voisins, ou k-NN pour k nearest neighbors, est simple. Étant donné un élément e dont on veut prédire la classe $c(e)$:

1. on sélectionne dans les données les k éléments de T les plus proches de e au sens de la distance d où k est un certain entier (dans un premier temps choisi de manière arbitraire) ;
2. on choisit comme prédiction de $c(e)$ la classe majoritaire parmi ces k voisins.

Sur le schéma suivant on représente un jeu d'élève positionné selon les coordonnées de leur domicile et répartis dans trois établissement (rouge, vert ou bleu) :



Supposons qu'un nouvel élève soit domicilié aux coordonnées $(0, 0)$. Voyons la prédiction de l'algorithme k-NN par exemple pour $k = 6$:



Parmi les 6 plus proches voisins du nouvel élève, 1 est bleu, 2 verts et 3 rouges. La prédiction choisie par l'algorithme sera donc rouge.

k-NN : Implémentation Nous allons maintenant implémenter l'algorithme k-NN de manière générale. On fait en Python les choix d'implémentation suivants :

— `T` est un tableau ;

— `d` est une fonction à deux arguments ;

— `c` est un dictionnaire.

Pour déterminer les k plus proches voisins du nouvel élément e on peut trier T par proximité avec e :

```
def knn(T, e, k, d):
    t = sorted(T, key = lambda elt : d(e, elt))
    return t[0:k]
```

Une fois ces plus proches voisins déterminés, il faut déterminer la classe la plus fréquente parmi eux :

```
def classe_plus_frequente(t, c):
    #on compte les occurrences de chaque element dans un dictionnaire
    occurrences = {}
    for elt in t:
        if c[elt] not in occurrences.keys():
            occurrences[c[elt]] = 1
        else:
            occurrences[c[elt]] += 1

    #on determine l element majoritaire et on renvoie sa classe
    c_maj = c[t[0]]
    for elt in t:
        if occurrences[c[elt]]>occurrences[c_maj]:
            c_maj = c[elt]
    return c_maj
```

On peut maintenant écrire notre fonction de prédiction.

```
def prediction_knn(T, c, e, k, d):
    t = knn(T, e, k, d)
    return classe_plus_frequente(t, c)
```

Cette implémentation s'adapte au problème traité à travers les différents arguments de la fonction.

1. T, e et c sont donnés par le problème ;
2. d et k doivent être choisis au mieux.

3 Choix de la distance

Jusqu'ici, on a utilisé le mot distance dans un sens intuitif mais on utilise en fait un notion bien définie mathématiquement. On appelle **pseudo-distance** (on dira distance par la suite) sur un ensemble E une fonction $d : E \times E \longrightarrow \mathbb{R}_+$ telle que :

— $\forall x \in E,\ d(x, x) = 0$

— $\forall x, y \in E,\ d(x, y) = d(y, x)$

— $\forall x, y, z \in E,\ d(x, z) \leq d(x, y) + d(y, z)$

Cette définition étant très large, il est possible de définir sur un ensemble de nombreuses distances différentes. C'est de ce choix que dépend essentiellement l'efficacité de l'algorithme k-NN.

Par exemple dans le plan, la distance couramment utilisée est **distance euclidienne** définie par :

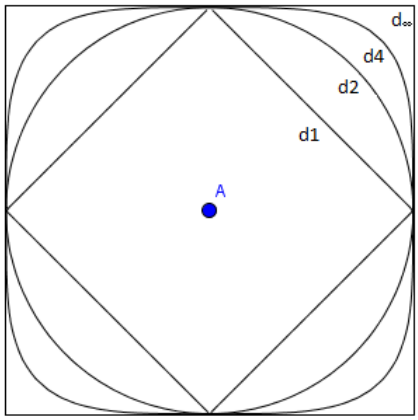
$$d_2(A,B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

```
def d_2(A, B):
    xA, yA = A
    xB, yB = B
    return sqrt((xB-xA)**2+(yB-yA)**2)
```

mais on peut vérifier que les définitions suivantes vérifient les mêmes propriétés :

$$\begin{aligned} d_1(A,B) &= |x_B - x_A| + |y_B - y_A| \\ d_4(A,B) &= \sqrt[4]{(x_B - x_A)^4 + (y_B - y_A)^4} \\ d_\infty(A,B) &= \max(|x_B - x_A|, |y_B - y_A|) \end{aligned}$$

On représente ci-dessous des **cercles** centrés en A selon la distance utilisée.



On peut également définir des distances entre des chaînes de caractères, par exemple :

- la **distance de Hamming** qui compte le nombre de caractères (éventuellement vides) deux à deux différents entre deux mots ;
- la **distance d'édition** qui compte le nombre minimum d'ajouts, de suppressions ou de modifications des lettres des deux mots pour les égaliser.

```
def d_hamming(s1, s2):
    res = 0
    for i in range(min(len(s1), len(s2))):
        s1[i]!=s2[i]:
            res+=1
    return res + abs(len(s1)-len(s2))
```

Afin de choisir une bonne notion de distance pour un algorithme des plus proches voisins, il faut choisir une distance pertinente vis à vis du problème. Même si ce choix est subjectif, il est primordial.

- **Exemple (carte scolaire) :** notre choix ne reposait que sur la distance géométrique entre les élèves. Si ce critère a certainement son importance, il n'est pas le seul à être pris en compte dans les faits. On aurait pu notamment prendre en compte les différentes options pratiquées par les élèves.
- **Exemple (l'algorithme de Youtube) :** Certains critères (abonnements, âge, etc.) peuvent être pertinents, d'autres (taille, localisation, prénom) ne le sont pas du tout.

4 Choix de k par validation croisée

Dans l'algorithme k-NN, on doit choisir le nombre k de voisins les plus proches que l'on regarde afin de déterminer une classe majoritaire. Se pose donc la question du choix de ce k dont vont dépendre les résultats de l'algorithme.

Une méthode de choix d'un tel k est la **validation croisée** :

1. On retire de T un certain nombre d'éléments ;
2. On fait tourner l'algorithme sur chacun de ces éléments en se servant de T', les données restantes pour différentes valeurs de k.
3. Puisqu'on connaît les classes des éléments testés, il est possible d'attribuer à chaque k un score (combien de fois l'algorithme a-t-il vu juste avec ce k ?). On choisit alors k ayant le meilleur score.

Voici un exemple fonction permettant de déterminer un k optimal.

```
def k_optimal(T, e, k, d, c):
    groupe_test = T[:20] #le 20 est ici arbitraire
    donnees_restantes = T[20:]
    k = 1
    score_max = 0
    for i in range(1, 11): #le 10 est ici arbitraire
        score = len([1 for e in groupe_test if prediction_knn(T, e, k, d, c) == c[e]])
        if score > score_max:
            score_max = score
            k = i
    return k
```

Attention : le k optimal ainsi calculé n'a encore une fois rien d'universel. Il dépend de la base de donnée dont on dispose et même potentiellement des éléments qui ont été extraits de T pour les tests.