Algorithme des k plus proches voisins (KNN)

Thibaut Cantaluppi

November 14, 2024

Algorithme d'apprentissage

Définition : Algorithme d'apprentissage supervisé

Inconnue : $f : X \longrightarrow Y$, où X un ensemble de **données** et Y un ensemble d'**étiquettes** (ou **classes**).
Entrée : des données d'**entraînement** $x_1, ..., x_n \in X$ et leurs étiquettes $f(x_1), ..., f(x_n) \in Y$.
Sortie : une fonction $g : X \longrightarrow Y$ approximant f.

A partir de données d'entraînement dont on connaît la classe, on veut prédire la classe de nouvelles données.

Algorithme d'apprentissage

Définition : Algorithme d'apprentissage supervisé

Inconnue : $f : X \longrightarrow Y$, où X un ensemble de **données** et Y un ensemble d'**étiquettes** (ou **classes**).
Entrée : des données d'**entraînement** $x_1, ..., x_n \in X$ et leurs étiquettes $f(x_1), ..., f(x_n) \in Y$.
Sortie : une fonction $g : X \longrightarrow Y$ approximant f.

A partir de données d'entraînement dont on connaît la classe, on veut prédire la classe de nouvelles données. Suivant l'ensemble possible d'étiquettes, on parle de :

- **Classification** : Y est fini, par exemple $Y = \{1, ..., k\}$.
  Exemples : k plus proches voisins (KNN classification), arbre de décision, réseau de neurones...

Algorithme d'apprentissage

Définition : Algorithme d'apprentissage supervisé

Inconnue : $f : X \longrightarrow Y$, où X un ensemble de **données** et Y un ensemble d'**étiquettes** (ou **classes**).
Entrée : des données d'**entraînement** $x_1, ..., x_n \in X$ et leurs étiquettes $f(x_1), ..., f(x_n) \in Y$.
Sortie : une fonction $g : X \longrightarrow Y$ approximant f.

A partir de données d'entraînement dont on connaît la classe, on veut prédire la classe de nouvelles données. Suivant l'ensemble possible d'étiquettes, on parle de :

- **Classification** : Y est fini, par exemple $Y = \{1, ..., k\}$.
  Exemples : k plus proches voisins (KNN classification), arbre de décision, réseau de neurones...
- **Régression** : Y est un ensemble continu, par exemple $Y = \mathbb{R}$.
  C'est donc une valeur dans Y que l'on veut prédire.
  Exemples : régression linéaire, GLM, KNN regression ...

Algorithme d'apprentissage

Définition : Algorithme d'apprentissage supervisé

Inconnue : $f : X \longrightarrow Y$, où X un ensemble de **données** et Y un ensemble d'**étiquettes** (ou **classes**).
Entrée : des données d'**entraînement** $x_1, ..., x_n \in X$ et leurs étiquettes $f(x_1), ..., f(x_n) \in Y$.
Sortie : une fonction $g : X \longrightarrow Y$ approximant f.

Exemples de problèmes de classification :

X	Y	$f(x)$
Tailles de tumeurs	Maligne, Bénigne	Quelle est la gravité de x ?
Mails	Spam, Non-spam	Est-ce que ce mail est un spam ?
Images	$[\![0, 9]\!]$	Quel chiffre est représenté sur x ?
Musiques	classique, rap, rock...	Genre musical de x

Algorithme des k plus proches voisins - Classification

Soit $k \in \mathbb{N}$.
L'algorithme des k plus proches voisins prédit la classe d'une nouvelle donnée x de la façon suivante :

1. Calculer les distances de x à toutes les données d'entraînement.

Algorithme des k plus proches voisins - Classification

Soit $k \in \mathbb{N}$.
L'algorithme des k plus proches voisins prédit la classe d'une nouvelle donnée x de la façon suivante :

1. Calculer les distances de x à toutes les données d'entraînement.
2. Trouver les k données d'entraînement les plus proches de x (en termes de distance).

Algorithme des k plus proches voisins - Classification

Soit $k \in \mathbb{N}$.
L'algorithme des k plus proches voisins prédit la classe d'une nouvelle donnée x de la façon suivante :

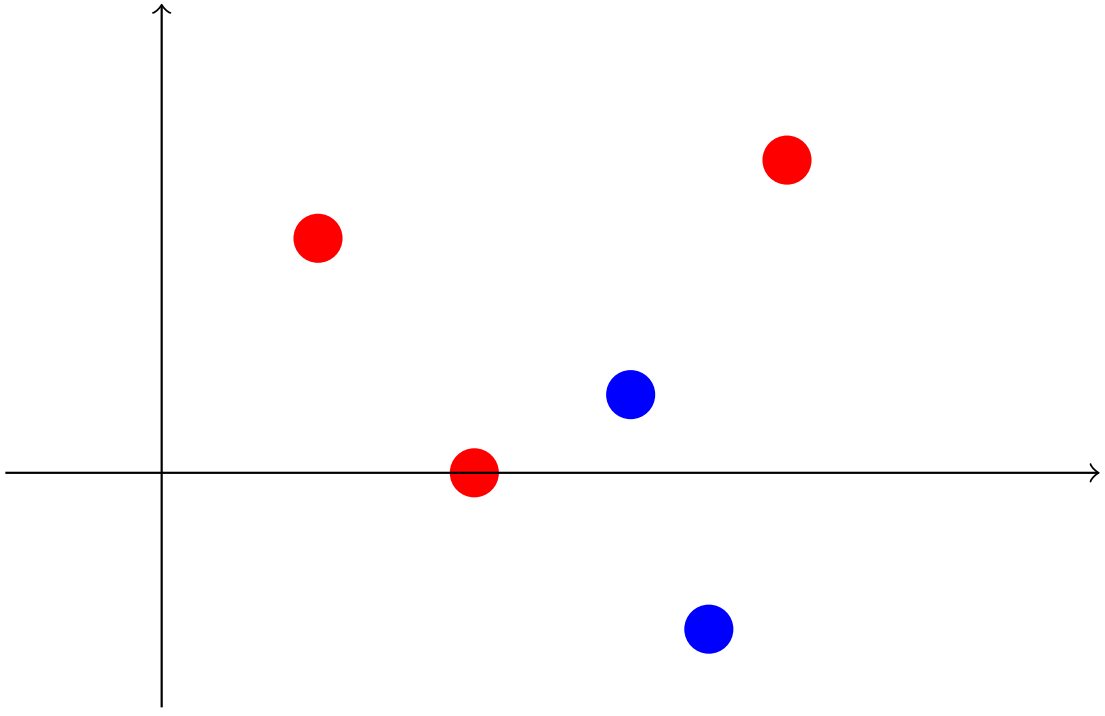
1. Calculer les distances de x à toutes les données d'entraînement.
2. Trouver les k données d'entraînement les plus proches de x (en termes de distance).
3. Trouver la classe majoritaire $c \in Y$ parmi ces k données les plus proches de x.

Algorithme des k plus proches voisins - Classification

Soit $k \in \mathbb{N}$.
L'algorithme des k plus proches voisins prédit la classe d'une nouvelle donnée x de la façon suivante :

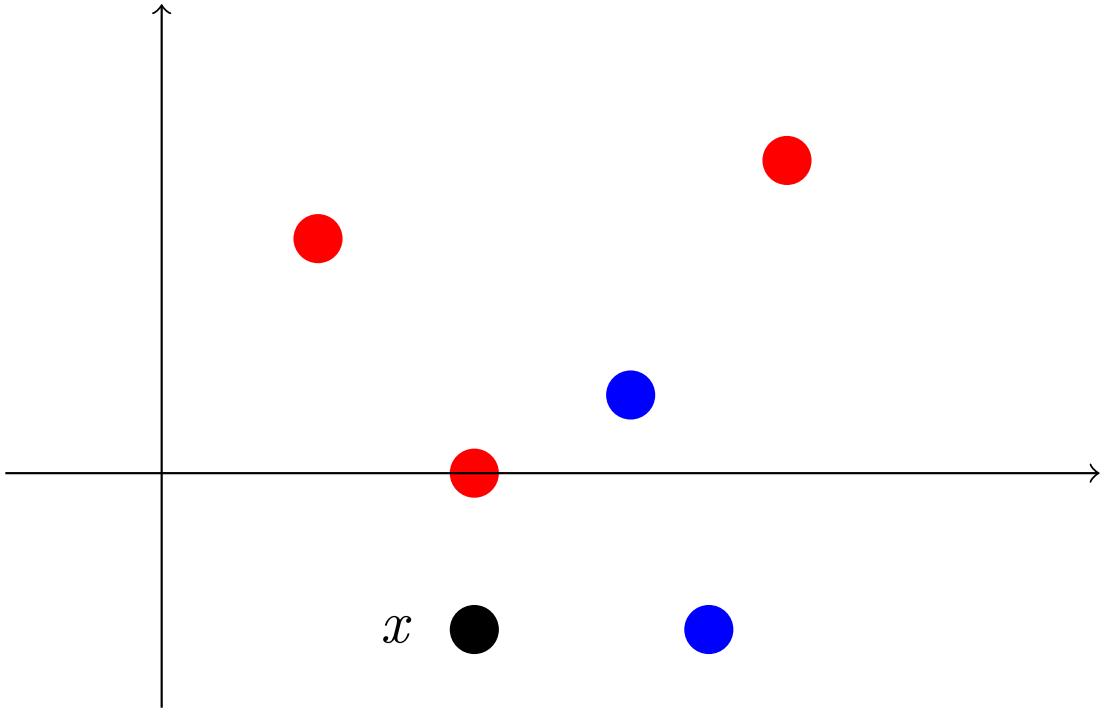
1. Calculer les distances de x à toutes les données d'entraînement.
2. Trouver les k données d'entraînement les plus proches de x (en termes de distance).
3. Trouver la classe majoritaire $c \in Y$ parmi ces k données les plus proches de x.
4. Prédire que x est de classe c.

Algorithme des k plus proches voisins - Classification



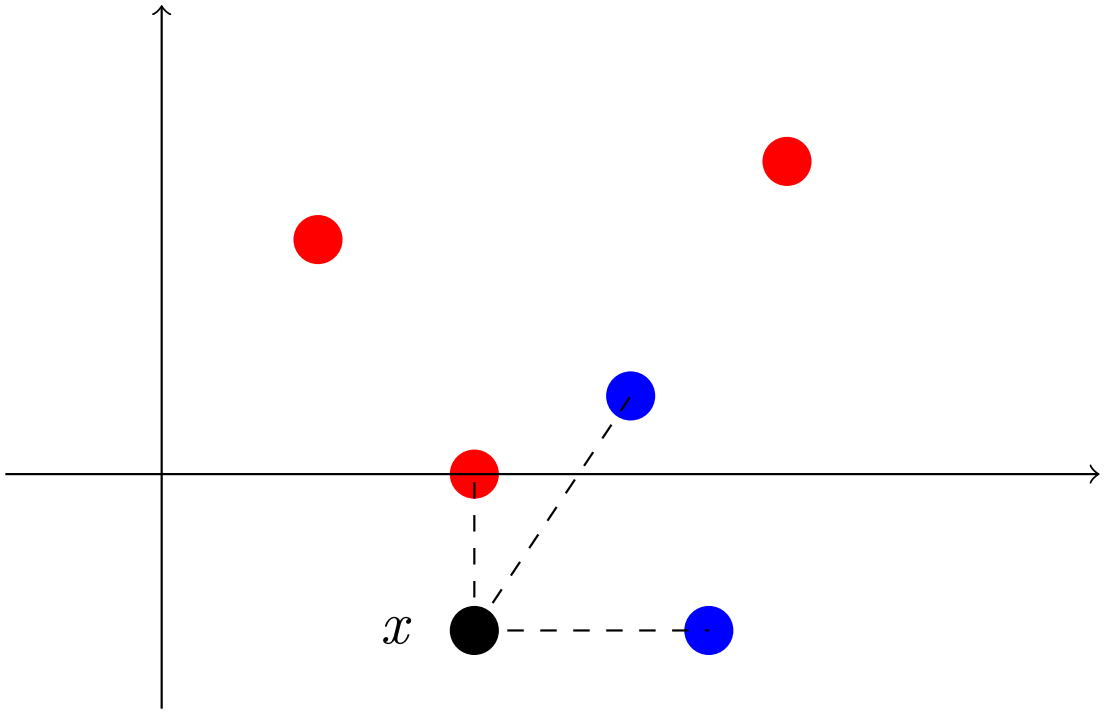
Des données dont les classes (rouge ou bleues) sont connues.

Algorithme des k plus proches voisins - Classification



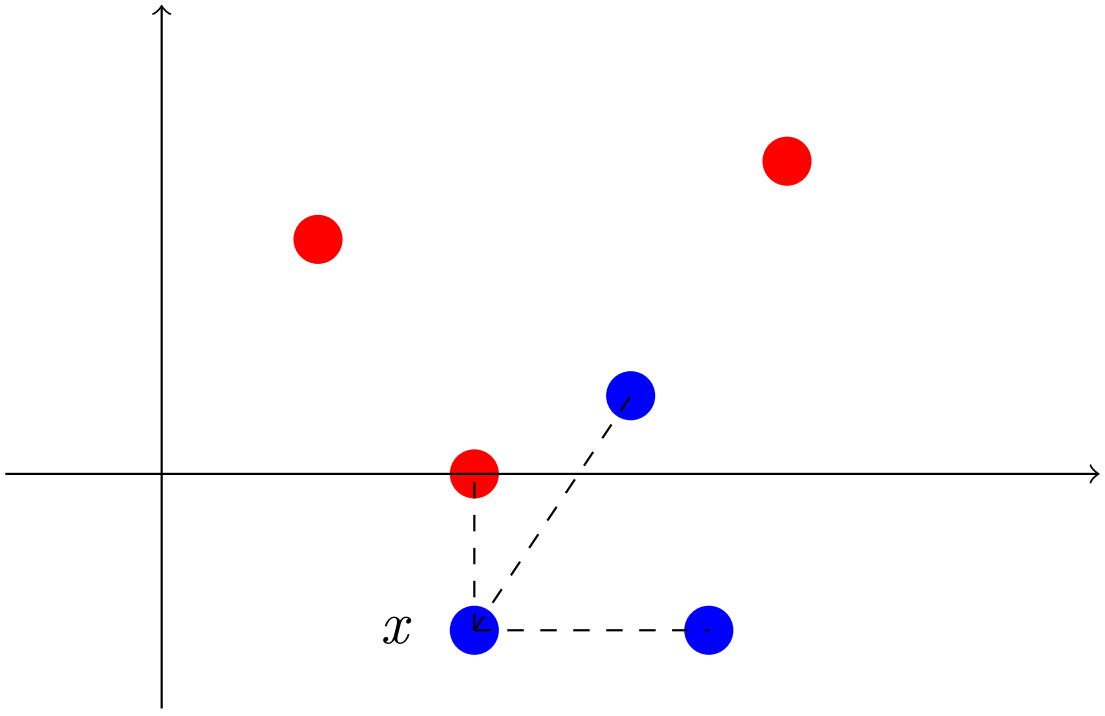
On veut prédire la classe d'une nouvelle donnée x.

Algorithme des k plus proches voisins - Classification



On trouve les k plus proches voisins. La valeur de k choisie à l'avance va impacter la décision prise par l'algorithme : k est un **hyperparamètre**.

Algorithme des k plus proches voisins - Classification



On associe à x la classe majoritaire de ses plus proches voisins.

Algorithme des k plus proches voisins - Classification

Question

Proposer d'autres hyperparamètres pour l'algorithme KNN.

Algorithme des k plus proches voisins - Classification

Question

Proposer d'autres hyperparamètres pour l'algorithme KNN.

La **métrique** utilisée pour caractériser la distance entre deux point (distance euclidienne, de Manhattan, de Minkowsky...).

La répartition des **poids** des voisins (uniforme, proportionnelle à la distance...).

Étape 1 : trouver les k plus proches voisins.

Question

Soit `x` un vecteur (sous forme de liste), `X` une liste de vecteurs, `k` un entier et `d` une fonction de distance.
Écrire une fonction `voisins(x, X, k)` renvoyant la liste des k plus proches voisins de `x` dans `X`.

Étape 1 : trouver les k plus proches voisins.

Question

Soit `x` un vecteur (sous forme de liste), `X` une liste de vecteurs, `k` un entier et `d` une fonction de distance.
Écrire une fonction `voisins(x, X, k)` renvoyant la liste des k plus proches voisins de `x` dans `X`.

```
def voisins(x, X, k):
    I = [] # indices des plus proches voisins de x
    for i in range(k): # ajout du ième minimum
        jmin = None
        for j in range(len(X)):
            if j not in I and (jmin is None or d(x, X[j]) < d(x, X
                jmin = j
        I.append(jmin)
    return I
```

Complexité : $O(kn(p+k))$, où $n = |X|$ et p est le coût de $d(x, y)$

Étape 1 : trouver les k plus proches voisins.

Autre solution : on peut aussi trier les données d'entraînement par ordre croissant de distance à x et prendre les k premières.

```
def voisins(x, X, k):
    # renvoie les k plus proches voisins de x dans X
    indices = sorted(range(len(X)), key=lambda i: d(x, X[i]))
    return indices[:k]
```

Complexité :

Étape 1 : trouver les k plus proches voisins.

Autre solution : on peut aussi trier les données d'entraînement par ordre croissant de distance à x et prendre les k premières.

```
def voisins(x, X, k):
    # renvoie les k plus proches voisins de x dans X
    indices = sorted(range(len(X)), key=lambda i: d(x, X[i]))
    return indices[:k]
```

Complexité : $\mathrm{O}(np + n\log(n))$.

Étape 1 : trouver les k plus proches voisins.

Autre solution (MP Option info) : On peut aussi utiliser une file de priorité max implémentée par un tas min avec le module `heapq` :

```
from heapq import heappush, heappushpop
def voisins(x, X, k):
    f = []
    for i in range(len(X)):
        c = (-d(x, X[i]), i)
        if len(f) < k:
            heappush(f, c)
        else:
            heappushpop(f, c) # push suivi d'un pop
    return [f[i][1] for i in range(k)]
```

Complexité :

Étape 1 : trouver les k plus proches voisins.

Autre solution (MP Option info) : On peut aussi utiliser une file de priorité max implémentée par un tas min avec le module `heapq` :

```
from heapq import heappush, heappushpop
def voisins(x, X, k):
    f = []
    for i in range(len(X)):
        c = (-d(x, X[i]), i)
        if len(f) < k:
            heappush(f, c)
        else:
            heappushpop(f, c) # push suivi d'un pop
    return [f[i][1] for i in range(k)]
```

Complexité : $O(np + n\log(k))$.

Étape 1 : trouver les k plus proches voisins.

Autre solution (MP Option info) : on peut aussi utiliser `heapify` pour transformer une liste en tas en $O(n)$:

```
from heapq import heapify
def voisins(x, X, k):
    f = [(-d(x, X[i]), X[i]) for i in range(len(X))]
    heapify(f) # O(n)
    V = []
    for i in range(k): # O(klog(n))
        V.append(heappop(f)[1])
    return V
```

Complexité : $O(np + k\log(n))$.

Étape 2 : trouver la classe majoritaire

Question

Écrire une fonction `maj(L)` renvoyant l'élément le plus fréquent de la liste `L`, en complexité linéaire.

Étape 2 : trouver la classe majoritaire

Question

Écrire une fonction `maj(L)` renvoyant l'élément le plus fréquent de la liste `L`, en complexité linéaire.

Version simple :

```
def maj(L):
    compte = {} # compte[e] = nombre d'occurrences de e dans L
    for e in L:
        if e in compte:
            compte[e] += 1
        else:
            compte[e] = 1
    kmax = L[0]
    for k in compte:
        if compte[k] > compte[kmax]:
            kmax = k
    return kmax
```

Étape 2 : trouver la classe majoritaire

Question

Écrire une fonction `maj(L)` renvoyant l'élément le plus fréquent de la liste `L`, en complexité linéaire.

Version courte :

```
def maj(L):
    compte = {}
    for e in L:
        compte[e] = compte.get(e, 0) + 1
    return max(compte, key=compte.get)
```

$dico.get(k, d)$ renvoie la valeur associée à la clé k dans le dictionnaire $dico$. Le paramètre d est optionnel et spécifie une valeur de retour par défaut dans le cas où la clé k n'existerait pas.

Étape 3 : prédire la classe de x

```
def knn(x, X, Y, k):
    """ Prédit la classe de x avec l'algorithme KNN
    x : nouvelle donnée
    X : données d'entraînement
    Y : étiquettes des données d'entraînement
    k : nombre de voisins à considérer
    """
    V = voisins(x, X, k)
    return maj([Y[i] for i in V])
```

Évaluation d'un modèle

Supposons que l'on possède des données X avec des étiquettes Y et qu'on veuille savoir si KNN est un bon classifieur pour ces données.

Pour cela, on partitionne les données en deux ensembles :

- Ensemble d'entraînement X_{train} (de classes Y_{train}) : données parmi lesquelles on va chercher les k plus proches voisins.
- Ensemble de test X_{test} (de classes Y_{test}) : données utilisées pour évaluer le modèle, en comparant les classes prédites par KNN avec les classes réelles.

Évaluation d'un modèle

```python
def separer(X, Y, p):
    """ Sépare les données X en 2 ensembles X_train et X_test
    p : pourcentage de données dans X_train
    """
    X_train, X_test, Y_train, Y_test = [], [], [], []
    for i in range(len(X)):
        if i < p * len(X):
            X_train.append(X[i])
            Y_train.append(Y[i])
        else:
            X_test.append(X[i])
            Y_test.append(Y[i])
    return X_train, X_test, Y_train, Y_test

def predict(x, k):
    return knn(x, X_train, Y_train, k)
```

Évaluation d'un modèle

Définition

- La **précision** d'un modèle est la proportion de données de test bien classées par rapport au nombre total de données.
- L'**erreur** est égale à 1 – précision.

```
def precision(k):
    n = len(X_test)
    p = 0
    for i in range(n):
        if predict(X_test[i], k) == Y_test[i]:
            p += 1
    return p/n
```

Évaluation d'un modèle

Définition

La **matrice de confusion** est une matrice carrée dont les lignes et les colonnes sont les classes possibles. La case (i, j) contient le nombre de données de test de classe i qui ont été prédites comme appartenant à la classe j.

Évaluation d'un modèle

Définition

La **matrice de confusion** est une matrice carrée dont les lignes et les colonnes sont les classes possibles. La case (i, j) contient le nombre de données de test de classe i qui ont été prédites comme appartenant à la classe j.

Exemple : Dans la matrice suivante, on voit que 21 données de classe 0 ont été prédites comme appartenant à la classe 0, 2 données de classe 1 ont été prédites comme appartenant à la classe 2...

```
array([[21,  0,  0],
       [ 0, 29,  1],
       [ 0,  2, 23]])
```

Question

Comment obtenir la précision à partir de la matrice de confusion ?

Évaluation d'un modèle

Question

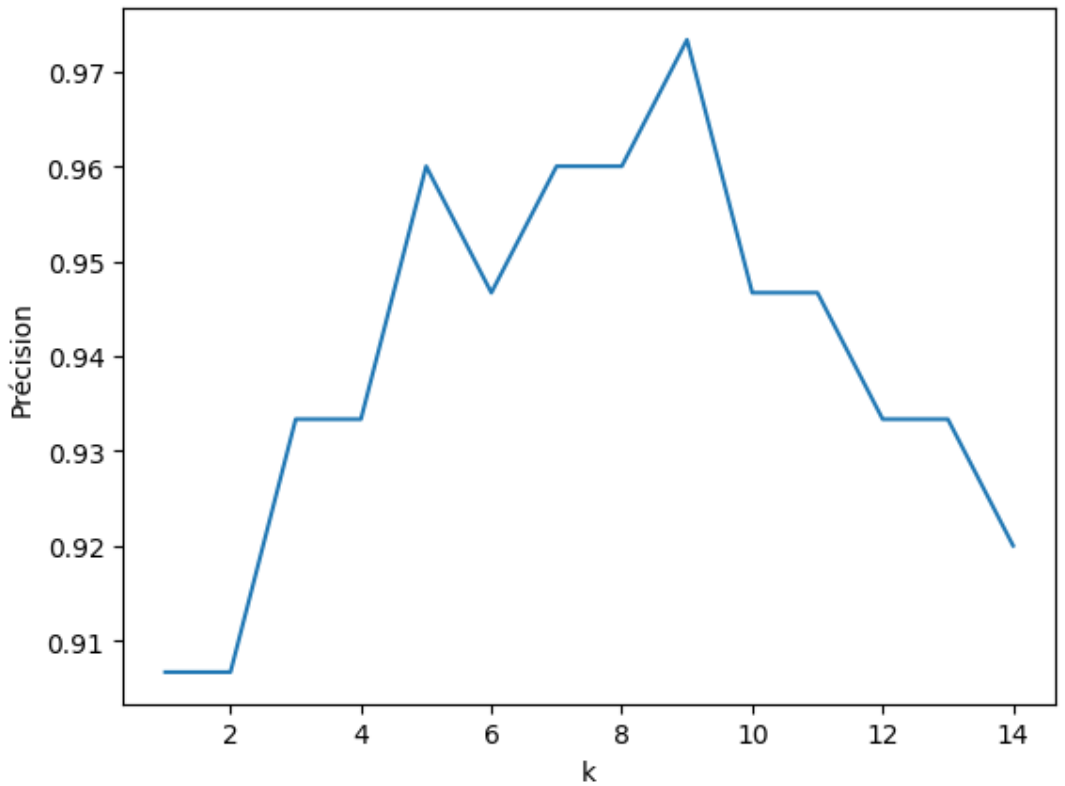
Comment choisir la valeur de k dans l'algorithme des k plus proches voisins ?

Évaluation d'un modèle

Question

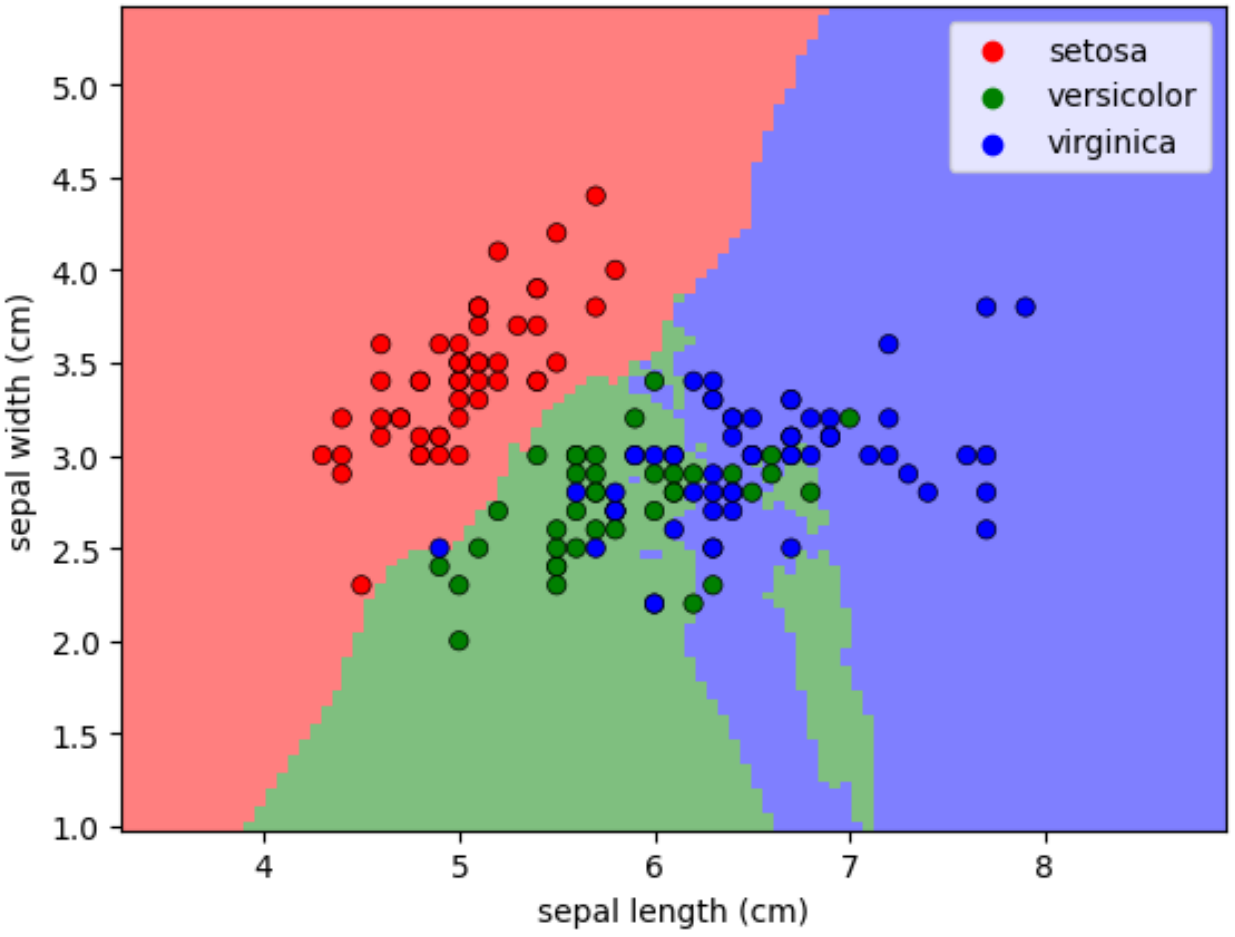
Comment choisir la valeur de k dans l'algorithme des k plus proches voisins ?

Réponse : on affiche la précision en fonction de k pour choisir la valeur de k qui donne la meilleure précision.



Évaluation d'un modèle

On peut aussi visualiser la **frontière de décision** (*decision boundary*) permettant de voir à quelle classe est associée chaque point de l'espace des données :



scikit-learn

scikit-learn est une bibliothèque Python qui contient de nombreux algorithmes d'apprentissage et fonctions utilitaires. Par exemple :

- `sklearn.neighbors.KNeighborsClassifier` : classification par les plus proches voisins, avec beaucoup d'options.
- `sklearn.model_selection.train_test_split` : séparer un ensemble de données en deux ensembles d'entraînement et de test.
- Pour plus d'informations, lire la documentation de scikit-learn.

Cependant, l'objectif de ce cours est plutôt de comprendre et réécrire les algorithmes plutôt que de les utiliser comme des boîtes noires.

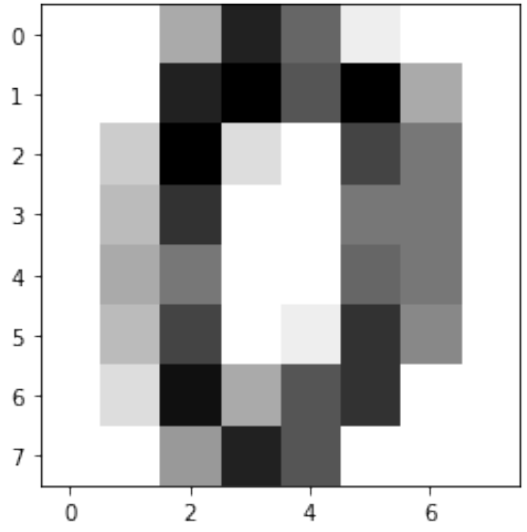
Exemple complet : classification d'iris

Autre exemple : classification de chiffres

MNIST est un jeu de données de chiffres manuscrits, utilisée initialement pour la reconnaissance de code postal. L'objectif est de reconnaître le chiffre écrit sur une image de taille 8×8.

Autre exemple : classification de chiffres

Chaque image est stockée sous forme d'une matrice 8×8 dont chaque élément est un niveau de gris entre 0 ($=$ blanc) et 15 ($=$ noir).



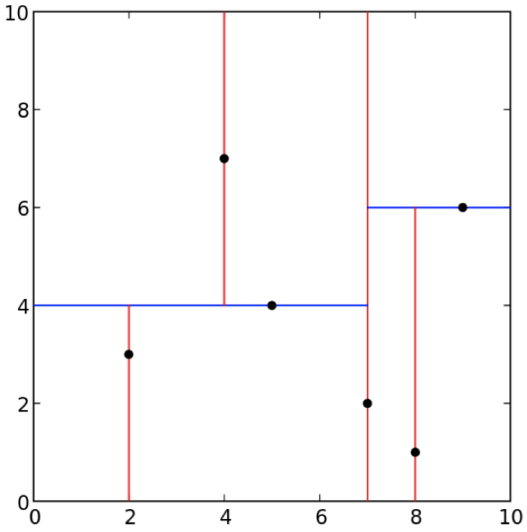

```
array([
       [ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

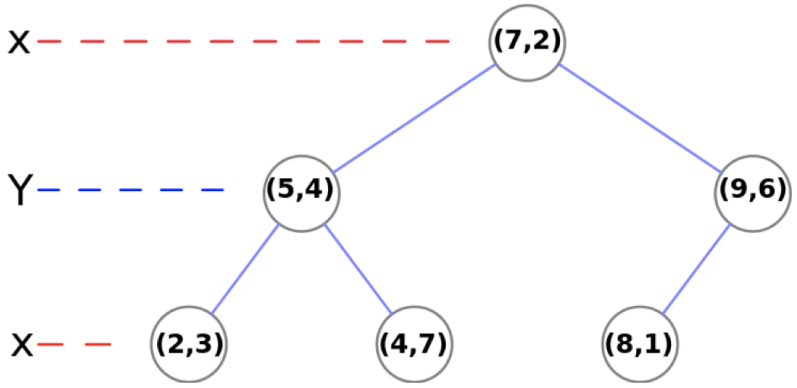
Autre exemple : classification de chiffres

On transforme une matrice à n lignes, p colonnes en un vecteur à $np = 64$ éléments.
On utilise, par exemple, la distance euclidienne sur $\mathbb{R}^{64}$ pour calculer la distance entre deux images.
Puis on applique la méthode des plus proches voisins pour prédire le chiffre sur une nouvelle image.
On obtient une précision d'environ 96%.

Complément : Arbre $k - d$

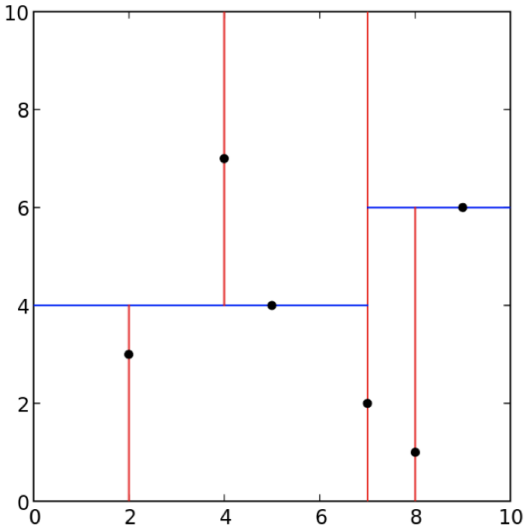
Un arbre $k - d$ est une structure de données permettant de calculer plus rapidement les plus proches voisins.

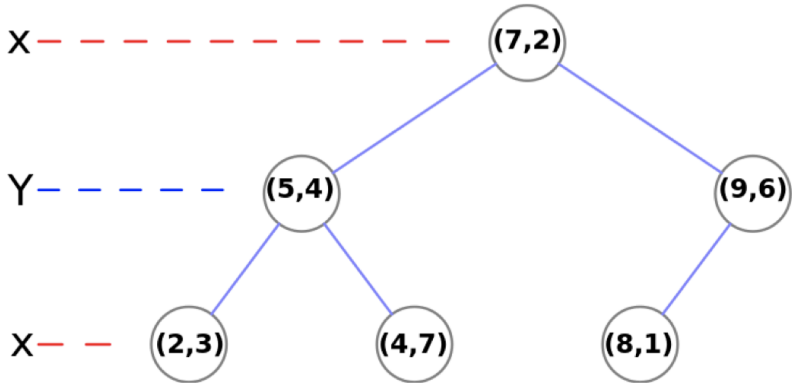




Complément : Arbre $k-d$

Un arbre $k-d$ est une structure de données permettant de calculer plus rapidement les plus proches voisins.

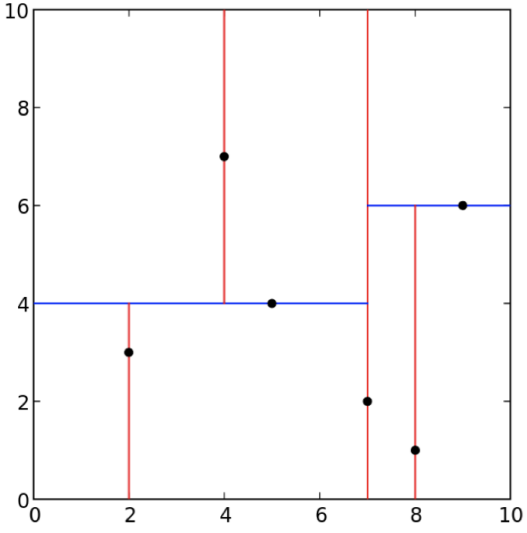


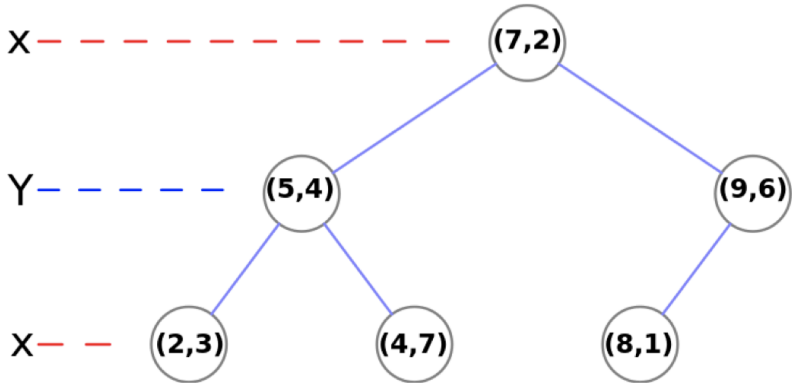


Construction de l'arbre : à la profondeur i, on divise les points de $\mathbb{R}^p$ en 2 parties en prenant comme axe de division l'axe i modulo p.

Complément : Arbre $k-d$

Un arbre $k-d$ est une structure de données permettant de calculer plus rapidement les plus proches voisins.





Construction de l'arbre : à la profondeur i, on divise les points de $\mathbb{R}^p$ en 2 parties en prenant comme axe de division l'axe i modulo p.
Recherche de plus proches voisin : on regarde en priorité le demi-espace contenant le point à chercher.
$\Longrightarrow$ $O(\log n)$ en moyenne pour trouver le plus proche voisin de x.