Apprentissage supervisé

ITC PC

M. Charles

Un peu de SQL

Exercice - CCINP 2024 (modifié)

Une base de données contient des informations sur un jeu, sur chaque joueur de ce jeu, ainsi que les parties effectuées entre les joueurs.

Table Joueur :

idJoueur	nom	prenom	niveau	naissance
...	...	...	...	...
18571	Martin	Jean	2048	23/02/1958
18572	Dupond	Marie	2103	03/01/1972
18573	Develion	Théo	1857	05/10/2004
...	...	...	...	...

Table Partie :

id_Partie	id_joueur1	id_joueur2	resultat	jour	jeu
1	1547	1568	0.5	08/01/2001	'egai...
2	1204	3	0	12/07/1998	'egaj...
3	4	2	1	15/07/2018	'egbi...
...	...	...	...	...	...

Exercice - CCINP 2024 (modifié)

Le schéma de la base de donnée est donné par :

```
Joueur(id_Joueur int, nom text, prenom text, niveau int, naissance date)
Partie(id_Partie int, id_joueur1 int, id_joueur2 int, resultat real, jour
date, jeu text)
```

Questions :

1. Écrire une requête SQL permettant d'extraire les identifiants des joueurs ayant un niveau strictement supérieur au score 1900.
2. Écrire une requête SQL permettant d'afficher le nom et le prénom des 3 joueurs ayant le niveau le plus élevé.
3. Écrire une requête SQL permettant de déterminer les joueurs ayant plus de cent victoires lorsqu'ils commencent la partie. La requête doit renvoyer le nom, le prénom et le nombre de victoires de ces joueurs classés par ordre décroissant du nombre de victoires.

Exercice - CCINP 2024 (modifié)

1. Écrire une requête SQL permettant d'extraire les identifiants des joueurs ayant un niveau strictement supérieur au score 1900.

   ```sql
   SELECT id_Joueur FROM Joueur WHERE niveau > 1900;
   ```

2. Écrire une requête SQL permettant d'afficher le nom et le prénom des 3 joueurs ayant le niveau le plus élevé.

   ```sql
   SELECT nom, prenom FROM Joueur LIMIT 3 ORDER BY niveau DESC;
   ```

3. Écrire une requête SQL permettant de déterminer les joueurs ayant plus de cent victoires lorsqu'ils commencent la partie. La requête doit renvoyer le nom, le prénom et le nombre de victoires de ces joueurs classés par ordre décroissant du nombre de victoires.

```sql
SELECT nom, prenom, COUNT(*) AS nb_victoires
FROM Joueur JOIN Partie ON id_Joueur = id_joueur1
GROUP BY id_Joueur
HAVING nb_victoires >= 100;
```

Remarque : l'opérateur `LIKE` est utilisé pour comparer des chaînes de caractères dans la clause `WHERE` des requêtes SQL. Ce mot-clé permet d'effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Le caractère _ (underscore) représente n'importe quel caractère, mais un seul caractère uniquement alors que le caractère pourcentage `%` peut être remplacé par un nombre quelconque (et possiblement nul) de caractères.

Par exemple parmi une recherche dans les communes de France, `nom LIKE '_ff%f%'` ne revoie que `Offendorf` alors que remplacer le _ par un `%` renvoie `Pfaffenhoffen` et `Staffelfelden` en plus de `Offendorf`.

4. Écrire une requête SQL permettant de déterminer le pourcentage de victoires du joueur1 pour les parties où la case d'indice `0` a été jouée en premier.

Remarque : l'opérateur `LIKE` est utilisé pour comparer des chaînes de caractères dans la clause `WHERE` des requêtes SQL. Ce mot-clé permet d'effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Le caractère _ (underscore) représente n'importe quel caractère, mais un seul caractère uniquement alors que le caractère pourcentage `%` peut être remplacé par un nombre quelconque (et possiblement nul) de caractères.

Par exemple parmi une recherche dans les communes de France, `nom LIKE '_ff%f%'` ne revoie que `Offendorf` alors que remplacer le _ par un `%` renvoie `Pfaffenhoffen` et `Staffelfelden` en plus de `Offendorf`.

4. Écrire une requête SQL permettant de déterminer le pourcentage de victoires du joueur1 pour les parties où la case d'indice `0` a été jouée en premier.

```sql
SELECT COUNT(*) / (SELECT COUNT(*) FROM Partie WHERE jeu LIKE 'a%')
FROM Partie WHERE resultat = 1 AND jeu LIKE 'a%';
```

IA, ML…

Intelligence artificielle (IA)

Domaine de recherche, toujours en cours de définition

- « *construction de programmes informatiques s'adonnant à des tâches qui [...] demandent des* ***processus mentaux de haut niveau tels que : l'apprentissage perceptuel, l'organisation de la mémoire et le raisonnement critique*** » (Marvin Lee Minsky)
- « *théorie et développement de systèmes informatiques capables d'accomplir des* ***tâches requérant normalement l'intelligence humaine, telles que la perception visuelle, la reconnaissance de la parole, la prise de décision, et la traduction entre langues***. » (Oxford English Dictionary)

Apprentissage automatique (ML)

Idée :

1. Partir d'un ensemble de données (les données d'entraînement) ;
2. en tirer un *modèle* permettant d'effectuer des prédictions ou des décisions.

Applications actuelles : Perception artificielle + aide à la décision

- Médecine
- Filtrage d'emails
- Reconnaissance de la parole
- Agriculture
- Vision artificielle

Apprentissage supervisé / non supervisé

- Supervisé : les données d'entrainement sont étiquetées (input et output)

  Exemple : reconnaissance d'images, analyse de sentiments, génération d'images avec un prompt, détection d'intrusions, etc.

- Non supervisé : notamment *data mining*, les données ne contiennent que l'input

  Exemple : classification automatique, LLM (dans une certaine mesure…), etc.

Plus précisément

- **Données :** vecteurs dans $\mathbb{R}^d$ (ou $[0,1]^d$).
- **Objectif :** structurer les données d'entrainement (par exemple, par *clustering*) et/ou prédire ou mimer, en l'automatisant, le comportement humain sur d'autres données similaires.

Classification supervisée : l'algorithme k-NN

Classification

Le but : prédire la classe d'un échantillon à partir d'exemples connus.

Exemple :

- on a des photos de chats et chiens, chaque photo étant étiquetée « chat » ou « chien » ;
- on veut entrainer l'ordinateur à distinguer la photo d'un chat de celle d'un chien.

L'humain le fait très bien (il l'a appris étant petit), mais on souhaite automatiser la tâche.

Les données étiquetées sont des éléments de $\mathbb{R}^d \times [\![1, r]\!]$ où r est le nombre d'étiquettes différentes.

Si (X, Y) est une donnée étiquetée,

- X est la valeur d'entrée,
- Y est la valeur de sortie : l'étiquette associée à X.

On souhaite

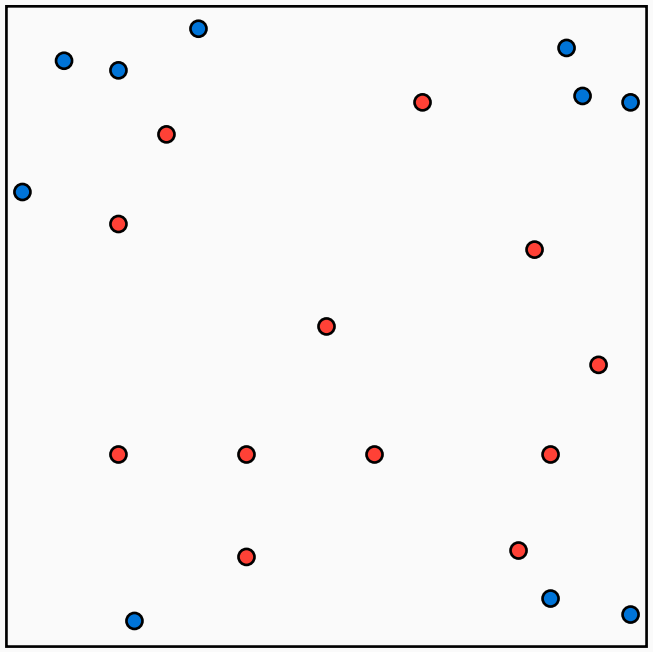
- en s'appuyant sur N données d'entrainement (X_n, Y_n),
- construire une application $f : \mathbb{R}^d \to [\![1, r]\!]$ telle que, si (X_n, Y_n) est une donnée déjà étiquetée, alors $f(X_n) = Y_n$.

L'algorithme k-NN

Si $X \in \mathbb{R}^d$, pour calculer $f(X)$:

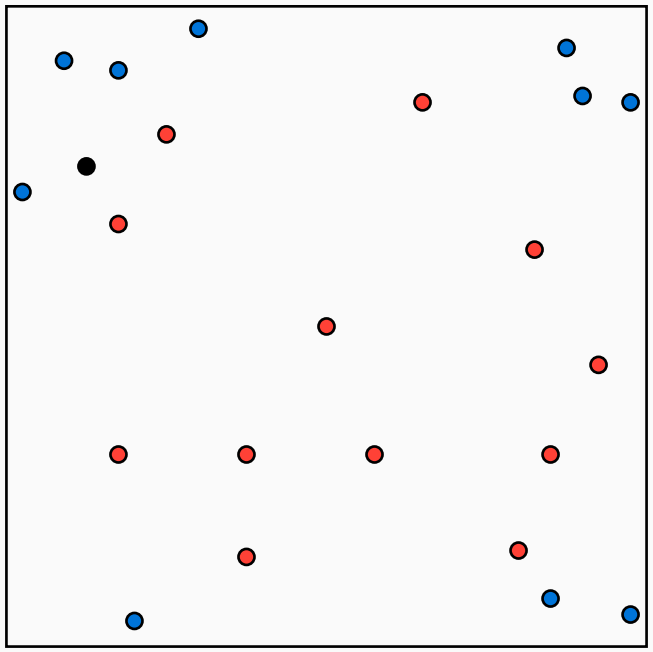
1. on fixe un entier strictement positif k ;
2. on cherche les k plus proches voisins de X parmi les données d'entrainement (pour la distance euclidienne dans $\mathbb{R}^d$) ;
3. parmi ces voisins, on cherche une étiquette de fréquence maximale ;
4. on définit $f(X)$ comme étant cette étiquette.

Exemple ($d = 2, r = 2$)



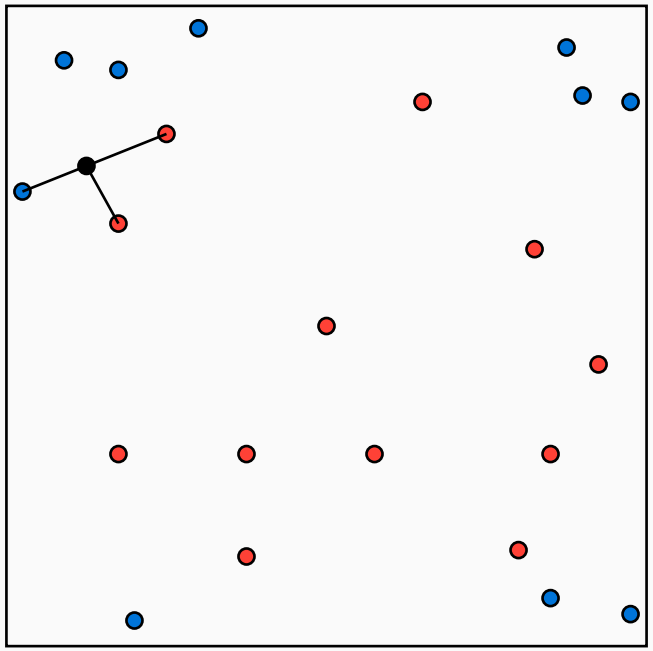
Données d'entrainement

Exemple ($d = 2, r = 2$)

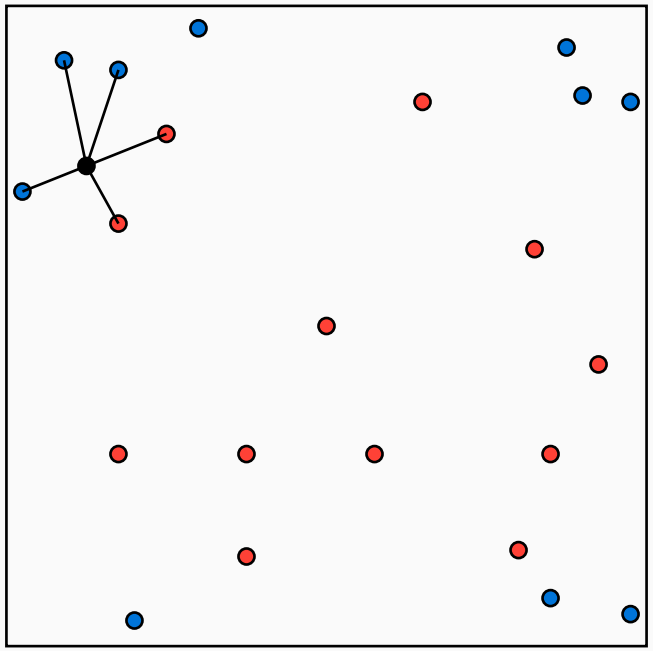


Données d'entrainement, et une donnée à classifier

Exemple ($d = r = 2$, $k = 3$)



Exemple ($d = r = 2$, $k = 5$)



Exercice

On suppose que les points d'entrées de $\mathbb{R}^d$ sont représentés par des listes de d flottants.

1. Écrire une fonction `dist2(p1: list, p2: list) -> float` qui renvoie le carré de la distance entre `p1` et `p2`.
2. Écrire une fonction `dist_min(p: list, data: list) -> int` qui renvoie l'indice du point le plus proche de `p` dans la liste `data`.
3. Adapter la fonction précédente pour qu'elle renvoie les deux points les plus proches.

Exercice

On suppose que les points d'entrées de $\mathbb{R}^d$ sont représentés par des listes de d flottants.

1. Écrire une fonction `dist2(p1: list, p2: list) -> float` qui renvoie le carré de la distance entre `p1` et `p2`.

```
def dist2(p1: list, p2: list) -> float:
    tot = 0.
    for i in range(len(p1)):
        delta = p1[i] - p2[i]
        tot += delta*delta
    return tot
```

2. Écrire une fonction `dist_min(p: list, data: list) -> int` qui renvoie l'indice du point le plus proche de `p` dans la liste `data`.
3. Adapter la fonction précédente pour qu'elle renvoie les deux points les plus proches.

Exercice

On suppose que les points d'entrées de $\mathbb{R}^d$ sont représentés par des listes de d flottants.

1. Écrire une fonction `dist2(p1: list, p2: list) -> float` qui renvoie le carré de la distance entre `p1` et `p2`.
2. Écrire une fonction `dist_min(p: list, data: list) -> int` qui renvoie l'indice du point le plus proche de `p` dans la liste `data`.

```
def dist_min(p: list, data: list) -> int:
    i_min = 0
    d2_min = dist2(p, data[i_min])
    for i in range(1, len(data)):
        d2 = dist2(p, data[i])
        if d2 < d2_min:
            d2_min = d2
            i_min = i
    return i_min
```

3. Adapter la fonction précédente pour qu'elle renvoie les deux points les plus proches.

Exercice

```
def dist_min2(p: list, data: list) -> (int,int):
    i_min1 = 0
    i_min2 = 1
    d2_min1 = dist2(p, data[i_min1])
    d2_min2 = dist2(p, data[i_min2])
    for i in range(1, len(data)):
        d2 = dist2(p, data[i])
        if d2 < d2_min1:
            d2_min2 = d2_min1
            i_min2 = i_min1
            d2_min1 = d2
            i_min1 = i
        elif d2 < d2_min2:
            d2_min2 = d2
            i_min2 = i
    return (i_min1, i_min2)
```

Paramétrage et évaluation de la méthode

Question du choix de k

Le choix de la valeur de k est un problème difficile :

- Pour $k = 1$, on sélectionne la classe du plus proche voisin.
  (Diagramme de Voronoï, ...)
- Plus k est grand,
  - plus le modèle est robuste,
  - moins les frontières entre les classes sont nettes,
  - plus l'algorithme est lent.
- Si $r = 2$ (deux classes), il peut être judicieux de se limiter aux paramètres k impairs pour éviter les ballottages !

Pour déterminer une valeur de k correcte, on peut faire ainsi :

1. on coupe les données d'entrainement en deux moitiés aléatoires : moitié A (qui vont servir à construire l'algorithme) et moitié B (qui vont servir à évaluer sa qualité) ;
2. on conserve les étiquettes de A et on cache celles de B ;
3. pour différentes valeurs de k, on effectue une mesure de la qualité de l'algorithme k-NN, construit à l'aide des données de A, et testé sur les données de B ;
4. on conserve la valeur de k qui maximise cette qualité.

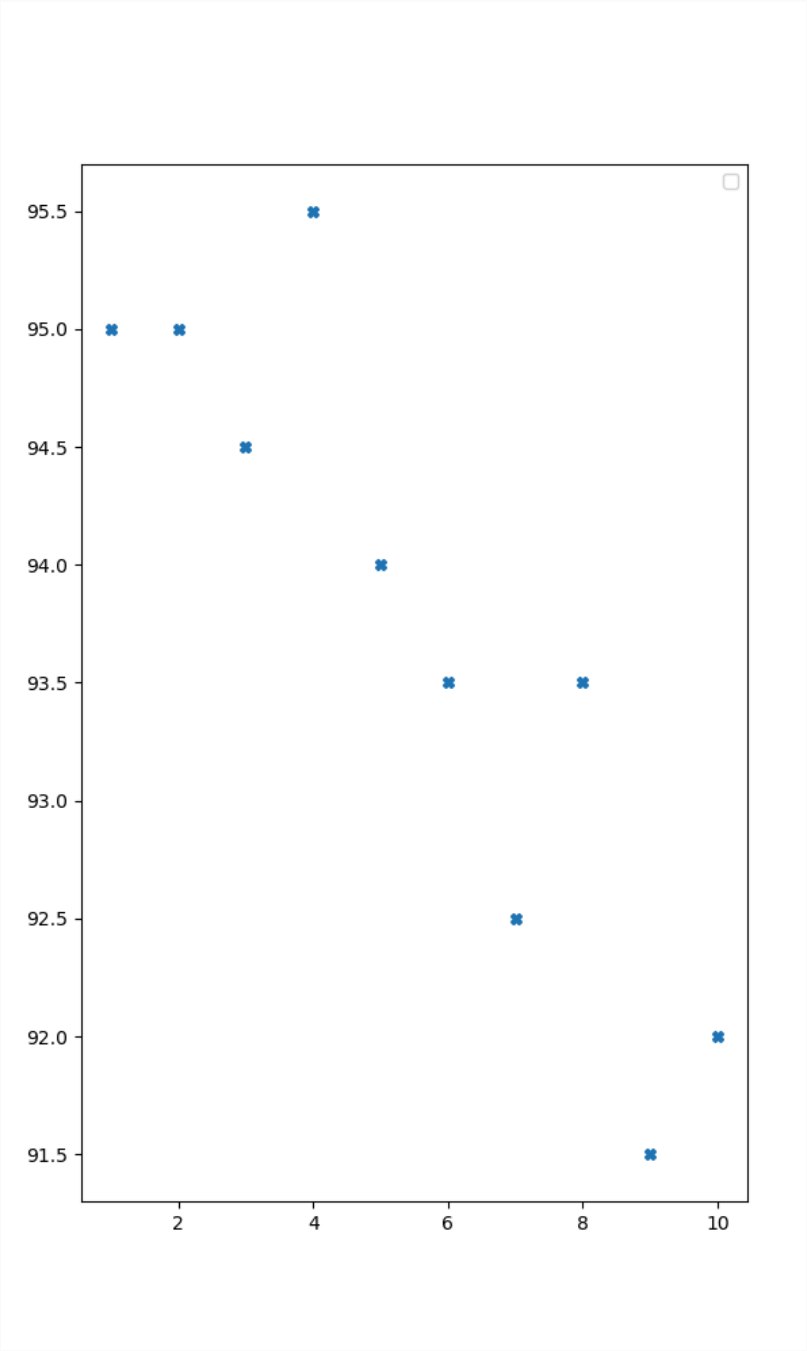
Par exemple, la qualité associée à une valeur de k peut être mesurée par la proportion d'étiquettes de B correctement retrouvées par l'algorithme à partir des données de A.

Exemple

Données d'entrainement : 400 points au hasard dans le carré $[-1, 1]^2$, en rouge ceux qui sont dans le disque unité, en bleu les autres.

On garde 50% de ces points en données-test, et on calcule le nombre de bonnes étiquettes pour k variant de 1 à 10.

Exemple



En abscisse : k, en ordonnée : la proportion d'étiquettes correctement reconnues.

Matrice de confusion

Pour évaluer plus finement la qualité d'un algorithme k-NN, on peut construire une matrice, dite matrice de confusion, dont le coefficient (i, j) est le nombre de données-test ayant l'étiquette i qui ont été classées par l'algorithme k-NN sous l'étiquette j.

Exemple pour $k = 4$

Avec le même exemple que précédemment ($k = 4$, étiquette 1 : bleu, étiquette 2 : rouge), on obtient par exemple

$$\begin{pmatrix} 28 & 6 \\ 3 & 163 \end{pmatrix}$$

ce que l'on interprète comme suit : parmi les données-test,

- 28 points bleus ont bien été détectés comme bleus,
- 6 points bleus ont été perçus comme rouges,
- 3 points rouges ont été perçus comme bleus,
- 163 points rouges ont bien été détectés comme rouges.

Exemples pour plusieurs valeurs de k

Voici les différentes matrices de confusion que l'on peut avoir sur l'exemple, en faisant varier k :

$$\begin{pmatrix} 28 & 6 \\ 4 & 162 \end{pmatrix} \begin{pmatrix} 28 & 6 \\ 4 & 162 \end{pmatrix} \begin{pmatrix} 25 & 9 \\ 2 & 164 \end{pmatrix} \begin{pmatrix} 28 & 6 \\ 3 & 163 \end{pmatrix} \begin{pmatrix} 23 & 11 \\ 1 & 165 \end{pmatrix}$$

$$k = 1 \qquad k = 2 \qquad k = 3 \qquad k = 4 \qquad k = 5$$

$$\begin{pmatrix} 24 & 10 \\ 3 & 163 \end{pmatrix} \begin{pmatrix} 23 & 11 \\ 4 & 162 \end{pmatrix} \begin{pmatrix} 24 & 10 \\ 3 & 163 \end{pmatrix} \begin{pmatrix} 19 & 15 \\ 2 & 164 \end{pmatrix} \begin{pmatrix} 21 & 13 \\ 3 & 163 \end{pmatrix}$$

$$k = 6 \qquad k = 7 \qquad k = 8 \qquad k = 9 \qquad k = 10$$

Résumé

La matrice de confusion peut aider à choisir la meilleure valeur de k. Elle est plus précise que le seul pourcentage de bonnes étiquettes (dans le cas d'une étiquette du type vrai-faux, elle apporte la distinction vrai-positif, faux-positif, vrai-négatif et faux-négatif).

Exercice

On suppose qu'on a un ensemble de données test sous la forme de liste `data` de couples de la forme `(p,label)`. On suppose que les étiquettes valent 0 ou 1.

On suppose également que l'on dispose d'une fonction `eval(p: list) -> int` qui associe à un point `p` une étiquette selon l'algorithme k-NN.

Écrire une fonction `mat_confusion(data: list)` qui renvoie la matrice `numpy` de confusion associée.

Exercice

On suppose qu'on a un ensemble de données test sous la forme de liste `data` de couples de la forme `(p,label)`. On suppose que les étiquettes valent 0 ou 1.

On suppose également que l'on dispose d'une fonction `eval(p: list) -> int` qui associe à un point `p` une étiquette selon l'algorithme k-NN.

Écrire une fonction `mat_confusion(data: list)` qui renvoie la matrice `numpy` de confusion associée.

```
import numpy as np
def mat_confusion(data: list):
    mat = np.zeros((2,2))
    for (p, label) in data:
        mat[(label, eval(p))] += 1
    return mat
```