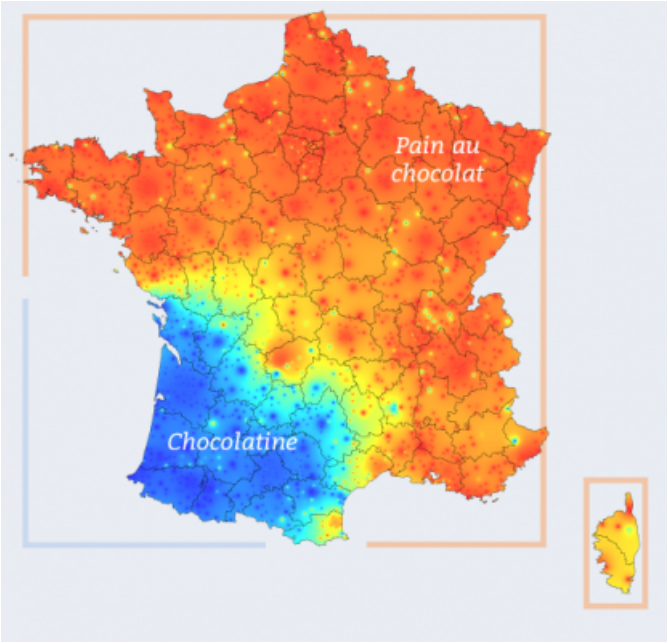
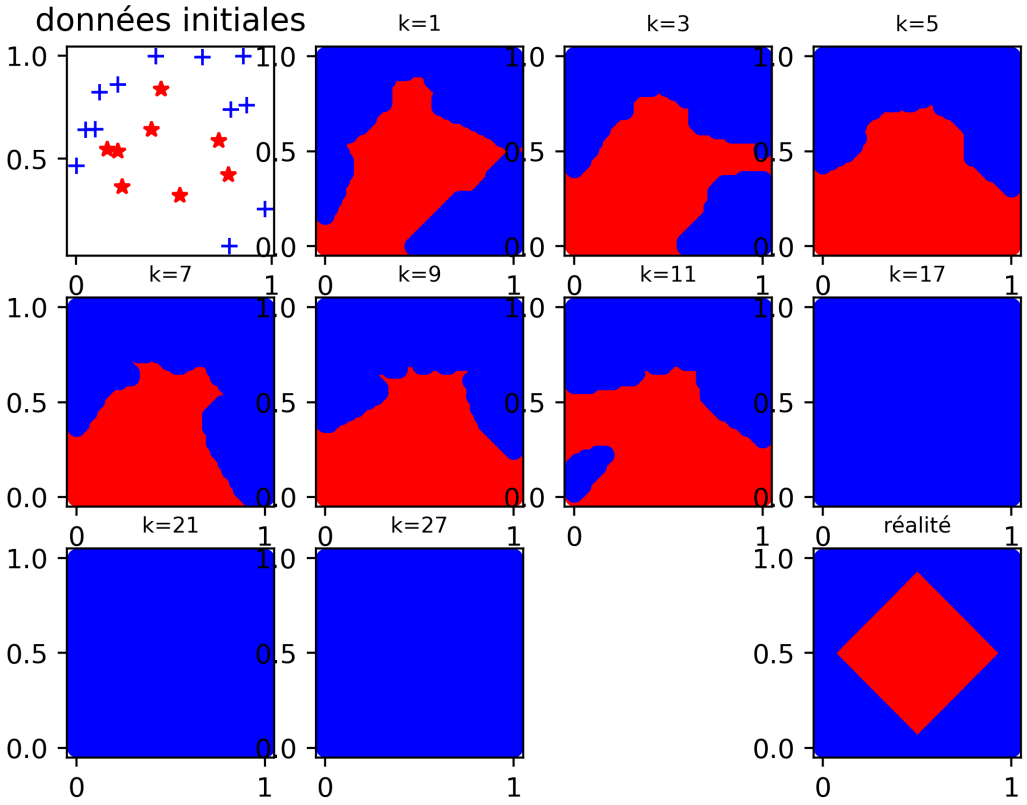
Algorithme des k plus proches voisins





Contexte

On dispose de données, chacune correspond à un individu possédant certains attributs (numériques) et une classe. Etant donné une nouvelle donnée et connaissant ses seuls attributs, peut-on prédire sa classe ?

1 Principe de la méthode et exemple

On dispose de points du plan $M_1, M_2, \cdots, M_n$ ayant chacun un label parmi les deux possibles : bleu et rouge. Ces n points sont en fait un **échantillon**, et on se propose à l'aide de ces données de prédire correctement

P. Connault – Reproduction et diffusion soumises à autorisation

la couleur de nouveaux points.

Soit P un nouveau point. Pour attribuer à P une couleur, on peut procéder ainsi :

1. Calculer les distances $PM_1, PM_2, \cdots, PM_n$
2. Retenir les k distances plus courtes, ainsi que les k points qui réalisent ces distances
3. Attribuer alors à P la couleur majoritaire parmi ces k points.

Cet algorithme est appelée la méthode des k **plus proches voisins**.

▲ DÉFINITION 1

Un algorithme basé sur les données, qui prédit (bien ou mal) la couleur de chaque point, est appelé un **estimateur**.

2 Implémentation en Python

2.1 Choix de représentation

Les données sont enregistrées dans une liste *donnees* contenant des triplets. Chaque triplet (abscisse, ordonnee, couleur) représente un point : ses coordonnées sont des flottants et sa couleur, 'rouge' ou 'bleu', est une chaîne de caractères.

On cherche à donner la couleur du point m grâce à la méthode des k plus proches voisins

2.2 Programme

```
def plusProchesVoisins(m, k, donnees):
    t = sorted(donnees, key=lambda d: (d[0] - m[0]) ** 2 + (d[1] - m[1]) ** 2)[:k]
#on trie le tableau donnees par distance au point m;  on garde les k plus proches sommets
    r=0 #nombre de rouges
    for d in t:
        if d[2] == "red":
            r += 1  #comptage rouge/bleu
    if 2 * r > k: #majorite rouge
        return "red"
    else:
        return "blue"
```

3 Exemple

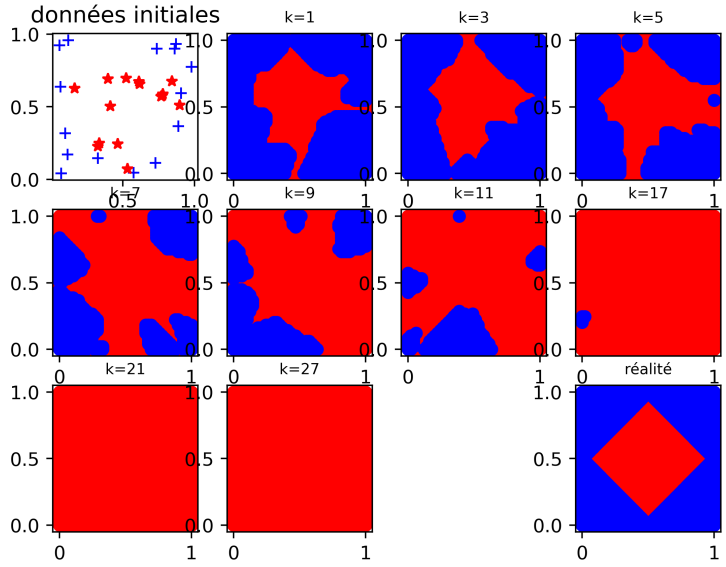
Dans les exemples, je choisis de donner :

1. La **réalité** en bas à droite (c'est-à-dire la bonne réponse pour chaque point) et un **échantillon** en haut à gauche (pour un certain nombre de points, la valeur observée).
   Le paradigme de l'apprentissage est d'estimer au mieux la réalité à partir de l'échantillon dont on dispose.
2. Ce que donne la méthode des k plus proches voisins, non pas pour un seul point mais pour chaque point du plan.
3. ...et ce pour différentes valeurs de k.
4. Enfin, les données peuvent être fausses : chaque point a une petite probabilité d'être mal classé (erreur de mesure, erreur de report...). Cette incertitude sur les données sera nommée le **bruit**.

Remarque

Plus formellement, on a le cadre suivant : on sait qu'il existe $f : P \longrightarrow \{R, B\}$ une certaine fonction f (la **réalité)**, et on connaît la valeur de f sur un ensemble fini A (ces valeurs forment l'**échantillon**).
On construit alors différents **estimateurs** de f, notés f_k où k prend différentes valeurs.
On veut trouver k tel que f et f_k coincident le mieux, c'est-à-dire : card $(\{x \in P \mid f(x) \neq f_k(x)\})$ est minimal.
Ici, j'assimile P a un ensemble fini pour que cela ait un sens.

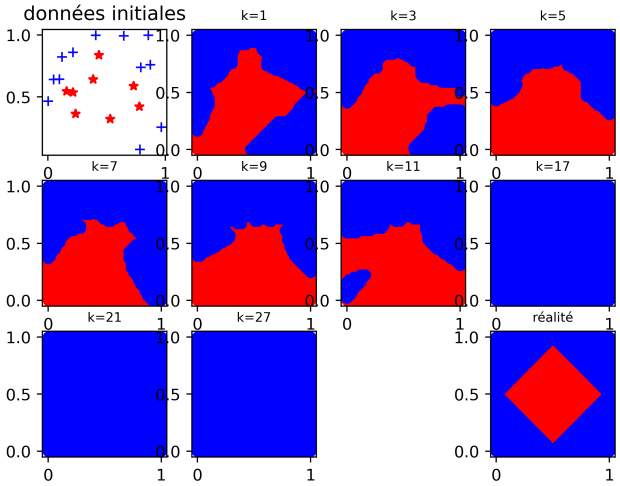
3.1 Bruit faible, trente données



Remarque

1 –pp voisins erreurs 368 matrice de confusion [[971, 229], [139, 1161]] 3 –pp voisins erreurs 353 matrice de confusion [[1027, 173], [180, 1120]] 5 –pp voisins erreurs 387 matrice de confusion [[1081, 119], [268, 1032]] 7 –pp voisins erreurs 698 matrice de confusion [[1173, 27], [671, 629]] 9 –pp voisins erreurs 881 matrice de confusion [[1151, 49], [832, 468]] 11 –pp voisins erreurs 1246 matrice de confusion [[1082, 118], [1128, 172]] 17 –pp voisins erreurs 1296 matrice de confusion [[1200, 0], [1296, 4]] 21 –pp voisins erreurs 1300 matrice de confusion [[1200, 0], [1300, 0]] 27 –pp voisins erreurs 1300 matrice de confusion [[1200, 0], [1300, 0]]

3.2 Bruit nul

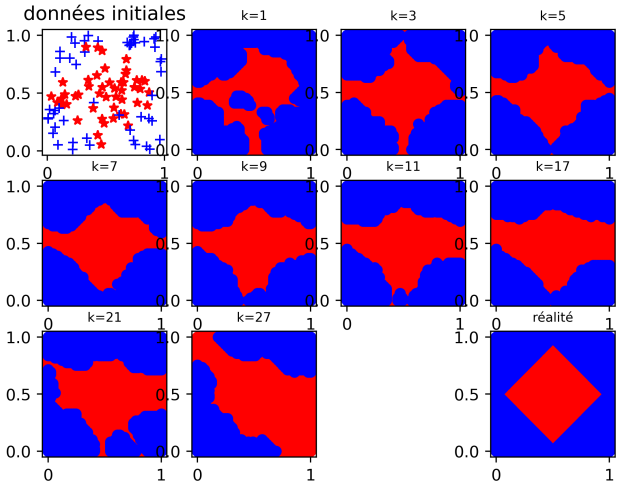


Remarque

1 –pp voisins erreurs 443 matrice de confusion [[1054, 146], [297, 1003]] 3 –pp voisins erreurs 578 matrice de confusion [[1008, 192], [386, 914]] 5 –pp voisins erreurs 920 matrice de confusion [[892, 308], [612, 688]] 7 –pp voisins erreurs 827 matrice de confusion [[815, 385], [442, 858]] 9 –pp voisins erreurs 982 matrice de confusion [[805, 395], [587, 713]] 11 –pp voisins erreurs 962 matrice de confusion [[821, 379], [583, 717]] 17 –pp voisins erreurs 1200 matrice de confusion [[0, 1200], [0, 1300]] 21 –pp voisins erreurs 1200 matrice de confusion [[0, 1200], [0, 1300]] 27 –pp voisins erreurs 1200 matrice de confusion [[0, 1200], [0, 1300]]

P. Connault – Reproduction et diffusion soumises à autorisation

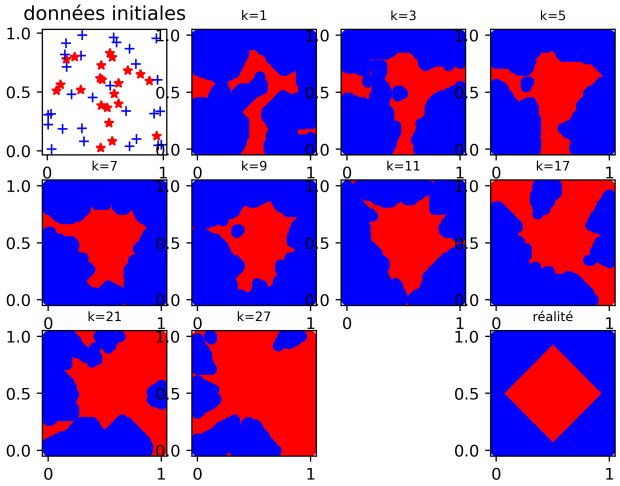
3.3 Bruit moyen



Remarque

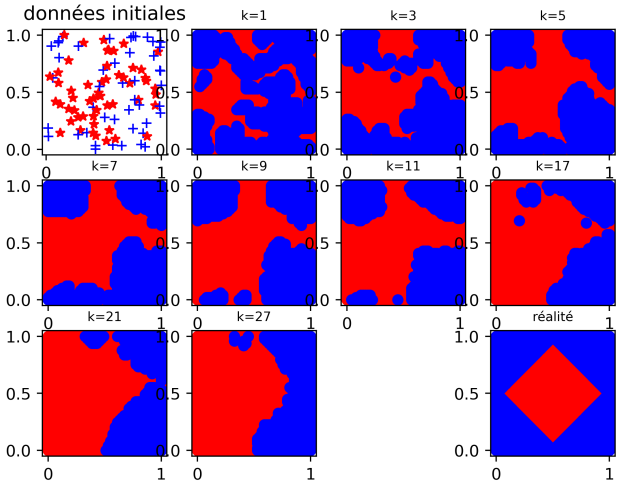
1 –pp voisins erreurs 313 matrice de confusion [[1047, 153], [160, 1140]] 3 –pp voisins erreurs 233 matrice de confusion [[1114, 86], [147, 1153]] 5 –pp voisins erreurs 204 matrice de confusion [[1110, 90], [114, 1186]] 7 –pp voisins erreurs 241 matrice de confusion [[1081, 119], [122, 1178]] 9 –pp voisins erreurs 312 matrice de confusion [[1094, 106], [206, 1094]] 11 –pp voisins erreurs 341 matrice de confusion [[1094, 106], [235, 1065]] 17 –pp voisins erreurs 337 matrice de confusion [[1097, 103], [234, 1066]] 21 –pp voisins erreurs 399 matrice de confusion [[1090, 110], [289, 1011]] 27 –pp voisins erreurs 670 matrice de confusion [[1093, 107], [563, 737]]

3.4 Bruit plus élevé



Remarque

1 –pp voisins erreurs 645 matrice de confusion [[752, 448], [197, 1103]] 3 –pp voisins erreurs 512 matrice de confusion [[897, 303], [209, 1091]] 5 –pp voisins erreurs 466 matrice de confusion [[941, 259], [207, 1093]] 7 –pp voisins erreurs 431 matrice de confusion [[914, 286], [145, 1155]] 9 –pp voisins erreurs 399 matrice de confusion [[948, 252], [147, 1153]] 11 –pp voisins erreurs 427 matrice de confusion [[1017, 183], [244, 1056]] 17 –pp voisins erreurs 925 matrice de confusion [[923, 277], [648, 652]] 21 –pp voisins erreurs 968 matrice de confusion [[951, 249], [719, 581]] 27 –pp voisins erreurs 1121 matrice de confusion [[1077, 123], [998, 302]]



Remarque

1 –pp voisins erreurs 845 matrice de confusion [[857, 343], [502, 798]] 3 –pp voisins erreurs 712 matrice de confusion [[972, 228], [484, 816]] 5 –pp voisins erreurs 683 matrice de confusion [[1026, 174], [509, 791]] 7 –pp voisins erreurs 636 matrice de confusion [[1054, 146], [490, 810]] 9 –pp voisins erreurs 679 matrice de confusion [[1062, 138], [541, 759]] 11 –pp voisins erreurs 642 matrice de confusion [[1083, 117], [525, 775]] 17 –pp voisins erreurs 855 matrice de confusion [[1067, 133], [722, 578]] 21 –pp voisins erreurs 863 matrice de confusion [[1069, 131], [732, 568]] 27 –pp voisins erreurs 816 matrice de confusion [[1067, 133], [683, 617]]

3.5 Code Python

```
1
2
3  def change_couleur(couleur):
4      if couleur=="red":
5          return "blue"
6      else:
7          return "red"
8
9
10 def ma_f(x,y): #cette fonction modelise la "realite" (inaccessible en pratique)
11     if abs(x-0.5)+abs(y-0.5)<=0.5:
12         return True
13     else:
14         return False
15
16
17
18 def genere_donnees(f, nbr=1000):
19     Cste=0 #niveau de bruit, a changer en fonction
20     donnees=[]
21     for _ in range(nbr):
22         x= random.random()
23         y= random.random()
24         if f(x,y):
25             couleur="red"
26         else:
27             couleur="blue"
28         if random.random()<Cste/(1+(abs(x-0.5)+abs(y-0.5)-0.5)**2):
29             couleur=change_couleur(couleur)
30             print((x,y))
31         # # variante avec bruit sans aspect geometrique
32         # p=0.2
33         # if random.random()<p:
34         #     couleur=change_couleur(couleur)
35         donnees.append((x,y,couleur))
36     return donnees
37
38 def matrice_confusion_sachant_f(donnees, f,k):
39     confusion=[[0,0],[0,0]]
40     for x in np.linspace(0,1,50):#a changer en fonction
41         for y in np.linspace(0,1,50):#idem
42             couleur_predite=plusProchesVoisins([x,y], k, donnees)
43             i,j=1,1 #par defaut reel=bleu et prediction=bleu
44             if f(x,y): #si la realite est rouge...
45                 i=0
46             if couleur_predite=="red": #si la prediction est rouge
47                 j=0
```

```
                confusion[i][j]+=1
    return confusion
```

4 Qualité de la prédiction : matrice de confusion, risque, calcul du risque

Pour évaluer la qualité de la prédiction, on se donne de **nouveaux** points.
On construit la méthode de prédiction sur des données d'apprentissage
On évalue la qualité de la méthode sur des données test où l'on connaît la réponse
Chaque nouveau point possède une classe « véritable » et une classe « prédite », on peut visualiser la qualité de l'estimateur grâce à la matrice de confusion.

4.1 Matrice de confusion, risque

▲ DÉFINITION 2

Soit un nouvel ensemble de N points. La **matrice de confusion** $(a_{i,j})$ contient en colonne i, ligne j, le nombre de points de couleurs i qui sont classés en couleur j par l'algorithme.

Remarque

1. Le nombre de points bien classés est $\operatorname{Tr} A$
2. Le nombre de points total est $N = \sum_{(i,j)\in[\![1,N]\!]^2} a_{i,j}$
3. Le nombre de points mal classés est $\sum_{(i,j)\in[\![1,N]\!]^2} a_{i,j} - \operatorname{Tr}(A)$
4. Le risque estimé de la méthode est $\dfrac{\sum_{(i,j)\in[\![1,N]\!]^2} a_{i,j} - \operatorname{Tr}(A)}{N} = \dfrac{N - \operatorname{Tr}(A)}{N}$

```
def matrice_confusion_sachant_f(donnees, f,k):
    confusion=[[0,0],[0,0]]
    for x in np.linspace(0,1,50):#50 est a changer en fonction de la precision voulue. Attention
        temps de calcul
        for y in np.linspace(0,1,50):#idem
            couleur_predite=plusProchesVoisins([x,y], k, donnees)
            i,j=1,1 #par defaut reel=bleu et prediction=bleu
            if f(x,y): #si la realite est rouge...
                i=0
            if couleur_predite=="red": #si la prediction est rouge
                j=0

            confusion[i][j]+=1
    return confusion
```

4.2 Choix du nombre k et compromis biais-variance

En fonction de la valeur choisie pour k, la valeur du risque théorique est différente : voici la valeur du risque $R(k)$ en fonction de k (moyenne sur 100 simulations).

```


def simulation(N,n, f):
    #N nbre de simuls, n nombre de points de donnees (points du plan) dans une simul
    Risque=np.zeros((N,9))
    for j in range(N):
        ddonnees=genere_donnees(f, n)
        for i in range(9):
            k=[1,3,5,7,9,11,17,21, 27][i]
            M=matrice_confusion_sachant_f(ddonnees, f,k)
            Risque[j,i]=M[0][1]+M[1][0]
    dico_stats={}
    for x in  [1,3,5,7,9,11,17,21, 27]:
        dico_stats[x]=0
    for k in (sorted(np.argmin(Risque,1))):
        x=[1,3,5,7,9,11,17,21, 27][k]
        dico_stats[x]+=1
    return Risque,dico_stats
```

On constate que ce risque est typiquement convexe avec un minimum « au milieu ». Les deux côtés extrêmes de la courbe de risque renvoient à deux phénomènes différents mais tous deux nocifs pour notre objectif de prédiction de la couleur correcte d'un point.

1. Pour k grand, on perd des détails de la figure. Par exemple, pour k égal au nombre de points, la couleur prédite est la même pour touts les points ! Le modèle choisi pour k trop grand a pris beaucoup de distance avec de la réalité car il en oubliant une part de l'information qu'apportent les données. On a un phénomène de **sous–apprentissage**, on dit aussi que le **biais** est trop élevé.
2. Pour k petit, c'est plus subtil. Dans la réalité, les données sont assez désordonnées : il y a des erreurs de mesure, des points abhérents... Si on prend $k = 1$ par exemple, on va tenir compte des erreurs de mesure sans les pondérer par leurs voisines. Le modèle retenu a comme « appris par coeur » les données en faisant du pointillisme. On parle alors d'**overfitting/overlearning/surapprentissage**, ou encore de **variance** trop élevée.

En apprentissage statistique, machine learning, intelligence articielle, un des enjeux est souvent de trouver un bon compromis entre biais et variance (« Bias–variance tradeoff ») c'est-à-dire d'éviter les deux écueils antagonistes que sont le sous-apprentissage et le surapprentissage.

Remarque
En cas d'overfitting le modèle retenu sera instable : une variation modérée de l'échantillon mènera à des fluctuations importantes de la prédiction. En cas de sous–apprentissage, on dispose généralement d'un modèle prédictif très stable et peu sensible aux données.

4.3 Danger : matrice de confusion calculée sur l'ensemble d'apprentissage

On peut calculer, à k fixé, la matrice de confusion A_k pour la méthode des k-plus-proches voisins en utilisant comme ensemble de référence **le même ensemble de points que celui ayant servi à construire la méthode**.

▲ DÉFINITION 3

Si $M_1, \cdots, M_n$ est un ensemble de points Le **risque empirique** $R(k)$ de la méthode des k-plus-proches voisins est défini comme le pourcentage de ces points qui sont mal classés par l'algorithme.
On a $R(k) = \dfrac{n - \mathrm{Tr}(A)}{n}$.

Remarque
Le risque empirique est globalement croissant en fonction de k, et est **nul** pour $k = 1$. En effet, le plus proche voisin de chaque point est lui-même.

Remarque
De même qu'on n'évalue pas une classe sur un énoncé déjà donné en cours, de même en intelligence artificielle, on n'évalue pas l'efficacité d'un apprentissage à partir des données d'apprentissage.
En particulier on n'assimile pas risque empirique et risque : le risque empirique minimise sur un apprentissage par coeur, mais ne cherche pas l'apprentissage « en prenant du recul par rapport au bruit ».

5 Choix de k pour un jeu de données réel : séparation en deux jeux de données ; validation croisée

En pratique on ne dispose pas de nouveaux points ! on peut alors les retirer d'avance du jeu de données et séparer nos données en un groupe **apprentissage** qui construira les différents estimateurs ; et un groupe **test**, qui nous permettra de choisir la valeur de k.

5.1 Séparation en deux parties du jeu de données

On coupe le jeu de données en une partie « jeu de données d'apprentissage » (2/3 des données) et une partie « jeu de données test » (1/3 des données).

Pour chaque valeur de k, on construit la méthode sur le jeu apprentissage, on estime le risque sur le jeu test, puis on retient le k minimisant ce risque estimé.

5.2 Validation croisée

On peut faire jouer ce jeu de rôle training/test aux différents morceaux du jeu de données et moyenner les résultats pour obtenir une estimation plsu robuste du risque.

5.2.1 N–fold cross–validation

On coupe le jeu de données en N parties de même taille. On mélange au besoin les données aupravant pour avoir N ensembles « représentatifs ».

```
Pour chaque k:
Risque[k]=0
Pour i=1 à N:
Le jeu test = le i eme morceau, le jeu d'apprentissage =les N-1 autres
Risque[k]+=(nombre de mal classés)
```

```
Prendre le k qui minimise Risque[k]
```

5.2.2 Leave One Out cross–validation (LOO)

```
Pour chaque k:
Risque[k]=0
Pour x in données:
Le jeu d'apprentissage = tout sauf x
Le jeu test =  une seule donnée: x
Risque[k]+=(x est mal classé)
Prendre le k qui minimise Risque[k]
```

5.2.3 Comparaison de plusieurs validations croisées

(bleu= 10-fold, vert= 5-fold, jaune=loo)

6 Solutions