Vision

Groupe Image

Membres du groupe

- Mariam Barhoumi mariam.barhoumi@u-psud.fr
- Leila-Yasmine Nedjar leila.nedjar@u-psud.fr
- Zakaria-Abderrahmen Difallah zakaria-abderrahmen.difallah@u-psud.fr
- Atte Torri atte.torri@u-psud.fr
- Matthieu Robeyns matthieu.robeyns@u-psud.fr
- Jean-Marc Fares jean-marc.fares@u-psud.fr

Page du challenge Vision
GitHub repository

Introduction

Contexte et motivation

L'actualité récente témoigne d'une part, de la faisabilité d'un système de conduite autonome sur un véhicule automobile [1-5], ainsi que d'un interêt non négligeable pour ce genre d'assistance à la conduite [Réf. nécessaire].
Afin de pouvoir envisager la réalisation d'un tel système, on se doit en premier lieu de pouvoir reconnaitre les différents obstacles qui seront présents sur le chemin du véhicule. Les données sont fournies par un capteur, généralement (comme pour [2,5]), il s'agît d'une caméra.

Problematique et Données

Lors de la conduite, un agent humain détecte et trie les stimulis (obstacles, panneaux, sons …) qu'il percoît pour les transformer en actions (accélération, freinage, changement de file, insertion …). Chacune des étapes est susceptible d'être acquise par un algorithme d'apprentissage [1, 2, 4]. Pour notre part, on se concentre sur l'interprétation des stimulis fournis par des capteurs, ici une caméra plus précisément. Nous nous donnons un jeu de données composé d'images représentant des objets ou obstacles pouvant être rencontrés sur une route : CIFAR10 [6]. Ce dernier se compose de 60'000 images colorées carrées de 32 pixels de côté réparties en 10 classes, soit 6'000 images par classes. Il comprend des images représentant des avions, des voitures, des oiseaux, des chats, des cerfs, des chiens, des grenouilles, des chevaux, des

bateaux et des camions. Le créateur nous fournit ces données dans des fichiers objet `cPickle` archivés. Pour pouvoir exploiter les données, nous utilisons la fonction `unpickle` proposée par l'auteur [Annexe 3]. Afin de mesurer l'efficacité du modèle au long de l'apperntissage, on utilise la métrique BAc (Balanced Accuracy) ; elle consiste en le calcul de la moyenne des précisions obtennues sur chaque classe (la précision est elle même calculée par le ratio des images bien classées sur le nombre d'images de la classe). Cette métrique est particulièrement adaptée à des ensembles de tests deséquilibrés, avec des nombres d'éléments par classe différents.

Axes de travail

Visualisation

La visualisation des données sert à mieux représenter les données brutes, pour permettre une lisibilité facile des données et voir plus clairement comment les données se séparent.

Plusieurs méthodes de visualisation existent, mais pour ce problème il est important d'utiliser une méthode qui réduit les dimensions, tout en conservant les structures complexes des données. Une première approche pourrait être à l'aide d'une méthode linéaire tel que le PCA, qui consiste à créer de nouveaux axes avec de nouvelles variables, qui sont le produit des colonnes des matrices. Cependant, cette méthode est très pratique pour peu de données, or ce n'est pas notre cas. C'est pour cela que les méthodes non-linéaires utiles pour la visualisation des données dans notre cas sont le t-SNE (déjà implanté par les M2), l'Isomap (déjà implanté par les M2) et le Sammon Mapping et peut-être d'autres.

La méthode t-SNE permet de réduire les dimensions en créant des nuages de points en respectant les distances entre chaque point dans le repère originel. Le problème de cette méthode est que sur une très grande dataset sa perpléxité (au sens de l'entropie de Sammon) peut atteindre des valeurs tel que 50. L'autre problème est que cette méthode utilise des formules quadratiques qui vont malheureusement écraser les petites variations de distances.

Le Sammon Mapping est une autre méthode non-linéaire, qui utilise les mêmes techniques que celles vues précédement mais elle minimise aussi les erreurs dites de Sammon en utilisant la descente de gradient. Le problème est que la descente de gradient est une méthode assez lente, car elle demande beaucoup d'itérations, et si les conditions initiales sont mal choisies, alors la méthohde devient obsolète.

L'Isomap est une méthode assez similaire aux deux autres, c'est-à-dire qu'elle va calculer les voisins de chaque point puis construire un graphe à l'aide de ces calculs. Mais, il va aussi utiliser d'autres algorithmes (par exemple l'algorithme de Dijkstra) afin d'éviter les noeuds dans les nuages de points en coupant les relations entre de trop lointains voisin. Ainsi, les plus courts chemins entre chaque voisin sont conservés afin d'éviter les "Manifold" (soit l'écrasement des données) pour enfin utiliser un algorithme d'échelle multidimensionnel (MDS) afin de faire un affichage claire, et lisible pour une meilleur visualisation des données.

Preprocessing

La phase de preprocessing consiste à réorganiser notre dataset en sélectionnant les données pertinentes [19], afin de l'alléger et permettre une classification plus efficace. Il existe pour cela différentes méthodes, dont la conformité [10] dépend du type de données à explorer. Pour notre part, comme il s'agit d'une classification multi-classe, l'apprentissage de nos données se fera dans des espaces de grandes dimensions, ce qui peut nous conduire à ce qu'on appelle un fléau de dimensionnalité [18]. On va alors commencer par procéder à une réduction de dimension. En effet, étant donné que notre base de données se résume à des images réparties en plusieurs classes, cela permettra de simplifier la visualisation de la structure du nuage de points qu'on aura obtenu à partir des variables (features) qu'on aura alors sélectionnées car jugées les mieux représentatives de l'information. Pour ce faire, on effectuera tout d'abord une sélection de variables (feature selection) dont la méthode [11] adoptée sera due au choix fait sur le critère de la pertinence du sous-ensemble de données considéré. On va pour cela, éliminer les données qui n'ont pas d'impact voire très peu sur la classification. Ainsi, si on retrouve deux features identiques ou présentant des valeurs à 0 celles-ci ne seront pas prises en compte.

Modelling

La modélisation prédictive est un processus qui utilise l'exploration de données et la probabilité pour prévoir les résultats. Chaque modèle est constitué d'un certain nombre de predicteurs [7], qui sont des variables susceptibles d'influencer les résultats futurs. Une fois que les données ont été recueillies pour les prédicteurs pertinents, un modèle statistique est formulé. Le modèle peut utiliser une équation linéaire simple ou d'un réseau neuronal complexe. À mesure que des données supplémentaires deviennent disponibles, le modèle d'analyse statistique est validé ou révisé.
Plusieurs algorithmes sont utilisés pour l'apprentissage supervisé plus précisément pour des problèmes de classification ce qui est le cas dans notre challenge. Les méthodes qu'on a comparées sont les suivantes :

Decision Tree :
Un arbre de décision est un modèle utilisé pour la classification, représenté sous forme d'un arbre dont chaque nœud représente le test de l'attribut, et on a une branche pour chaque valeur possible de l'attribut testé, puis à la fin on a les feuilles qui spécifient la classe de la variable [Annexe 4]. À la base les arbres de décision ont été développé pour le traitement des données qualitatives ce qui veut dire des variables avec des valeurs discrètes et non pas numériques, ils sont aussi connues par leur bon fonctionnement mais sur des données avec un petit nombre de caractéristiques (256 features dans notre cas c'est un peu trop).

Random Forest :
cet algorithme se base sur un ensemble d'arbre de décision (forêt), chaque arbre apprend sur un sous ensemble choisit aléatoirement, puis en faisant un vote majoritaire sur les classes choisies par chaque arbre on prédit la classe de la variable, pour mieux comprendre l'algorithme voir[15].

Naive Bayes :
Le Naive bayes est un modèle de probabilité conditionnel [14]: donné une instance de problème à classer, représentée par un vecteur x = (X1, … , Xn) représentant quelques n caractéristiques (variables

indépendantes), il affecte à cette instance des probabilités p(Ck ׀ X1 , … , Xn) pour chacun des K résultats ou classes Ck possibles[Annexe 5].

Neural Network Classifier :
Un réseau de neurones est constitué d'unités (neurones), disposées en couches, qui convertissent un vecteur d'entrée en une sortie. Chaque unité prend une entrée, lui applique une fonction (souvent non linéaire) puis transmet la sortie à la couche suivante. En règle générale, les réseaux sont définis comme feed-forward: une unité transmet sa sortie à toutes les unités de la couche suivante, mais il n'y a pas de retour sur la couche précédente[16]. Des pondérations sont appliquées aux signaux passant d'une unité à l'autre, et ce sont ces pondérations qui sont accordées dans la phase d'apprentissage pour adapter un réseau de neurones au problème particulier en question[Annexe 6].

k-Nearest Neighbour classification :
est une une méthode non paramétrique utilisée pour la classification [8]. L'entrée se compose des k exemples d'apprentissage les plus proches dans l'espace des caractéristiques et la sortie est une appartenance à une classe. Un objet est classé par un vote majoritaire de ses voisins, l'objet étant assigné à la classe la plus commune parmi ses k plus proches voisins (k est un entier positif, typiquement petit). Si k = 1, l'objet est simplement affecté à la classe de ce voisin le plus proche.

À mesure que la taille de l'ensemble de données d'apprentissage approche l'infini, cette méthode garantit un taux d'erreur inférieur au double du taux d'erreur minimal réalisable étant donné la distribution des données[Annexe 7].

Choix de la méthode On a trouvé plusieurs relations entre le "KNN", "decision tree" et le "random forest" et en voyant la complexité de leurs algorithmes on a opté pour notre challenge, qui est un problème de classification, d'utiliser l'algorithme du k-nearest neighbor (K-NN).

Resultats préliminaires

Annexes

Bibliographie

[1] : Mariusz Bojarski et al. End-to-End Learning for Self-Driving Cars. arXiv:1604, 2016. arXiv:1604.07316.
[2] : Lex Fridman. End-to-End Learning from Tesla Autopilot Driving Data. GitHub : lexfridman/deeptesla.
[3] : Shai Shalev-Shwartz, Shaked Shammah, Amnon Shashua. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. arXiv:1610, 2016. arXiv:1610.03295.
[4] : NVIDIA. The AI Car Computer for Autonomous Driving. NVIDIA's website.
[5] : Michael G. Bechtel et al. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. arXiv:1712, 2018. arXiv:1712.08644.
[6] : CIFAR10, Alex Krizhevsky, 2009. Learning Multiple Layers of Features from Tiny Images.
[7] : Margaret Rouse. Predictive Modeling. definition/predictive-modeling.

[8] : Altman, N.S.(1992). _"An introduction to kernel and nearest-neighbor nonparametric regression" Tandfonline: _ /doi/abs/10.1080/00031305.1992.10475879.
[9] : div3125. k-nearest-neighbors. GitHub : div3125/k-nearest-neighbors.
[10] : Claude Alain Saby. Study of Dimensionality Reduction Methods for Data Visualization. Résumé.
[11] : I. Guyon and A. Elisseeff. An introduction to variable and feature selection. Journal of Machine Learning Research, 2003. Abstract.
[12] : Laurens J.P. van der Maaten et Hinton, G.E.. _Visualizing High-Dimensional Data Using t-SNE _. Journal of Machine Learning Research, vol. 9, novembre 2008. vandermaaten08a.
[13] : tompollard. SammonMapping. Github: tompollard/sammon.
[14] : Narasimha Murty, Susheela Devi, (2011). Pattern Recognition: An Algorithmic Approach.
[15] : Saimadhu Polamuri,2017. How the random forest algorithme works in machine learning.
[16] : Aurélien Decelle, cours de vie artificielle S3 2017-2018.
[17] : tompollard. SammonMapping. Github: tompollard/sammon.
[18] : Richard Bellman. Dynamic programming. Princeton University Press, 1957. Explication sur OpenClassroom Qu'est-ce que le fléau de la dimension ?.
[20] : Balazs Feil & Janos Abonyi. Illustration for Sammon mapping. ResearchGate: (https://www.researchgate.net/figure/Illustration-for-Sammon-mapping_fig10_247930864)

- **Annexe 1** Table 1 [Statistiques]

Dataset	# of Examples	# of features	Sparsity	Categorical Variables	Missing data ?	# examples in class
Training	40000	256	59%	-	0%	[99 100 80 98 105 94 102 117 111 94]
Validation	10000	256	59%	-	0%	[107 102 109 91 105 99 112 79 101 95]
Test	10000	256	59%	-	0%	[102 88 103 118 74 115 112 94 94 100]

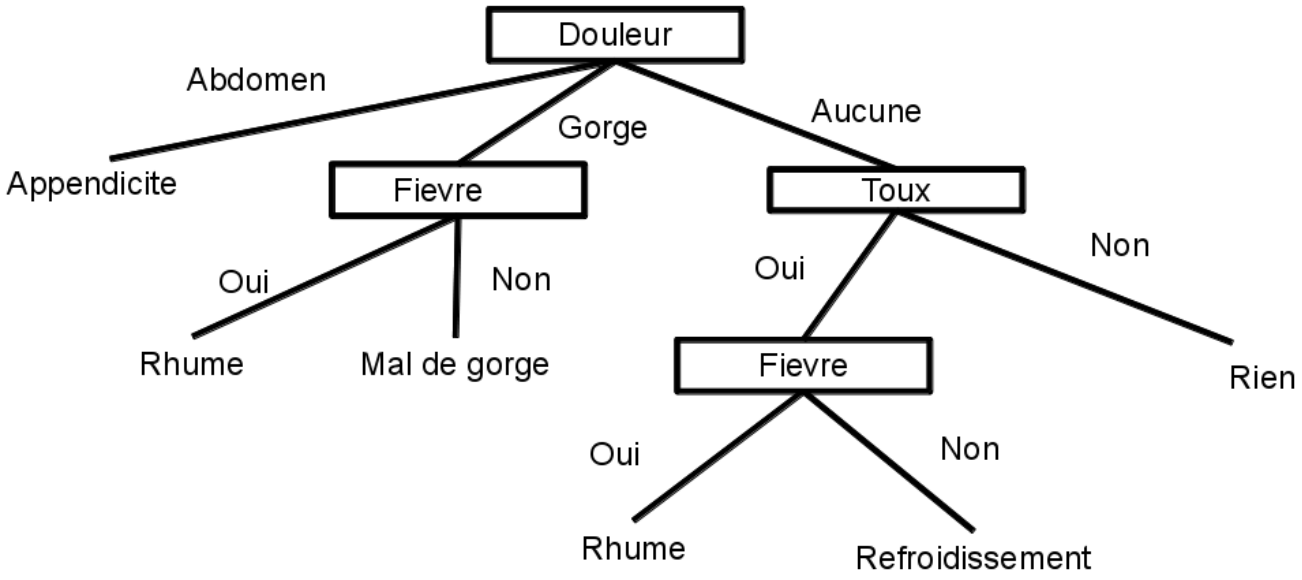
- **Annexe 2** Table 2 [Résultats préliminaires]

Method	k-NN	Neural Network	Random Forest	Decision tree	Naive Bayes
Traning	-	-	-	-	-
CV	-	-	-	-	-
Validation	-	-	-	-	-

- **Annexe 3** Methode de desarchivage

```
def unpickle(file):
    import cPickle
    with open(file, 'rb') as fo:
        dict = cPickle.load(fo)
    return dict
```

- **Annexe 4**
  Un exemple d'un arbre de décision pour la classification des malades:



- **Annexe 5** Voici un exemple de Naive bayes :

Multnomial Naive Bayes: Example

	docID	words in document	in c = China?
Training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
Test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$P(c) = \frac{3}{4} \qquad P(\bar{c}) = \frac{1}{4}$$

$$P(\text{Chinese}|c) = \frac{(5+1)}{(8+6)} = \frac{6}{14} = \frac{3}{7} \qquad P(\text{Toyko}|c) = P(\text{Japan}|c) = \frac{(0+1)}{(8+6)} = \frac{1}{14}$$

$$P(\text{Chinese}|\bar{c}) = \frac{(1+1)}{(3+6)} = \frac{2}{9} \qquad P(\text{Toyko}|\bar{c}) = P(\text{Japan}|\bar{c}) = \frac{(1+1)}{(3+6)} = \frac{2}{9}$$

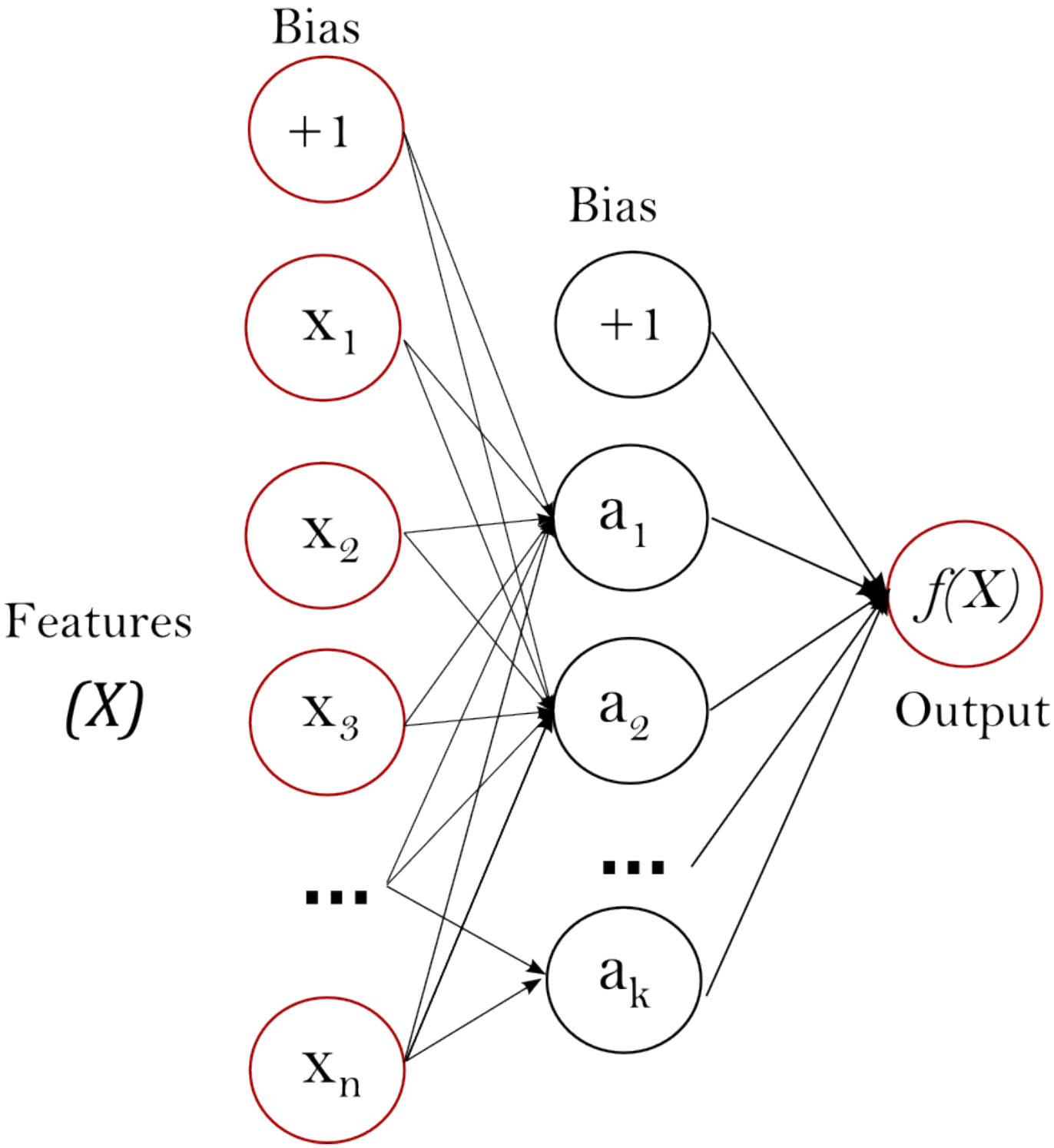
Pour savoir si le test set est dans c = china il suffit d'ajouter les calculs suivants :

- $P(class|doc) = P(c) \cdot \prod_{t=t_1}^{t_n} P(t_1|c)$
- $P(\overline{class}|doc) = P(\bar{c}) \cdot \prod_{t=t_1}^{t_n} P(t_1|\bar{c})$

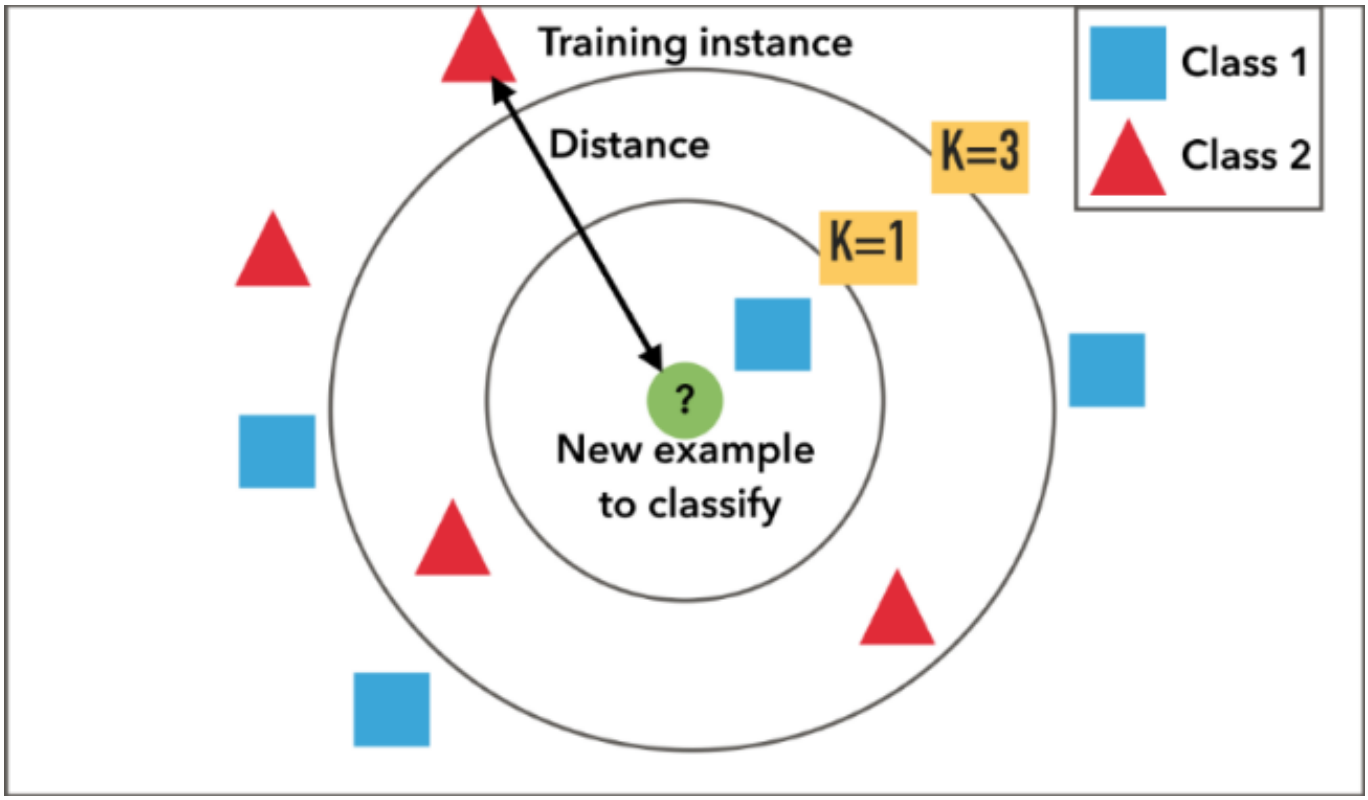
  Ceci pour une classe c et d(doc) qui contient les termes t1,t2,…,tn.

- **Annexe 6**

  Un exemple de perceptron avec une seule couche cachée:



- **Annexe 7**
  Voici un exemple de K-NN pour mieux comprendre cette méthode:



On a ici deux classes d'éléments (carrés bleus et triangles rouges), et nous essayons de placer un nouvel élément (représenté par un cercle vert),l'élement inconnu, dans l'une de ces deux classes, on va regarder les voisins les plus proches et voter par exemple pour k = 1, le nouvel exemple est classé comme classe 1, pour k = 3, le nouvel exemple serait placé dans la classe 2 et ainsi de suite jusqu'à trouver la classe à laquelle appartient cet élément.

- **Annexe 8** [13] : Pseudo-code du Sammon mapping (pour voir du code de T-SNE ou Isomap aller regarder le code des Masters2)

```
"""
  @param init : choisie la méthode linéaire "PCA" ou "Random"
  @param x : La base de donnée choisie
  @param inputdist : Verifie si les données sont bruts ou si elles sont déjà
  @param n : Nombre de dimension pour la visualisation
  @param display : permet un affichage de la progression du programme
  @param maxhalves : Plafonne le nombre d'étape pour la méthode step-halve
  @param maxiter : Plafonne le nombre d'itération possible pour éviter un "Ma
  @param tolfun : Erreur relative à notre fonction
"""
import numpy as np
from scipy.spatial.distance import cdist

def sammon(x, n = 2, display = 2, inputdist = 'raw', maxhalves = 20, maxiter

   X = x

  # Create distance matrix unless given by parameters
```

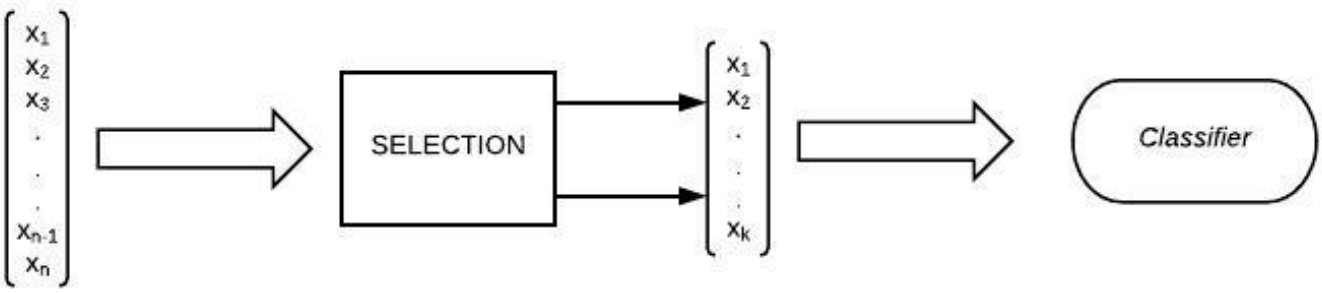
```
    # xD is the matrix with all distance
    if inputdist == 'distance':
        xD = X
    else:
        xD = cdist(X, X)

    # Remaining initialisation
    N = X.shape[0] # hmmm, shape[1]?
    scale = 0.5 / xD.sum()

    if init == 'pca':
        [UU,DD,_] = np.linalg.svd(X)
        Y = UU[:,:n]*DD[:n]
    else:
        Y = np.random.normal(0.0,1.0,[N,n])

    # Create unit matrix
    one = np.ones([N,n])

    # We create the inverse of distance and fix exception
    xD = xD + np.eye(N)
    xDinv = 1 / xD # Returns inf where D = 0.
    xDinv[np.isinf(xDinv)] = 0 # Fix by replacing inf with 0 (default Matlab be
    yD = cdist(Y, Y) + np.eye(N)
    yDinv = 1. / yD # Returns inf where d = 0.

    xDinv[np.isnan(xDinv)] = 0
    yDinv[np.isnan(xDinv)] = 0
    xDinv[np.isinf(xDinv)] = 0
    yDinv[np.isinf(yDinv)] = 0 # Fix by replacing inf with 0 (default Matlab be

    # Create Sammon's stress
    delta = xD - yD
    E = ((delta**2)*xDinv).sum()

    # Beggin of progressby gradient descent and step-halving procedure
    for i in range(maxiter):

        # Compute gradient, Hessian and search direction (note it is actually
        # 1/4 of the gradient and Hessian, but the step size is just the ratio
        # of the gradient and the diagonal of the Hessian so it doesn't
        # matter).
        delta = yDinv - xDinv
        deltaone = np.dot(delta,one)
        g = np.dot(delta, Y) - (Y * deltaone)
        dinv3 = yDinv ** 3
        y2 = Y ** 2
        H = np.dot(dinv3,y2) - deltaone - np.dot(2, Y) * np.dot(dinv3, Y) + y2
```

```
        s = -g.flatten(order='F') / np.abs(H.flatten(order='F'))
        y_old = Y

        # Use step-halving procedure to ensure progress is made
        for j in range(maxhalves):
            s_reshape = s.reshape(2,len(s)/2).T
            y = y_old + s_reshape
            d = cdist(y, y) + np.eye(N)
            dinv = 1 / d # Returns inf where D = 0.
            dinv[np.isinf(dinv)] = 0 # Fix by replacing inf with 0 (default Mat
            delta = xD - d
            E_new = ((delta**2)*xDinv).sum()
            if E_new < E:
                break
            else:
                s = np.dot(0.5,s)

        # Bomb out if too many halving steps are required
        if j == maxhalves:
            print('Warning: maxhalves exceeded. Sammon mapping may not converge

        # Evaluate termination criterion
        if np.abs((E - E_new) / E) < tolfun:
            if display:
                print('TolFun exceeded: Optimisation terminated')
            break

        # Report progress
        E = E_new
        if display > 1:
            print('epoch = ' + str(i) + ': E = ' + str(E * scale))

    # Fiddle stress to match the original Sammon paper
    E = E * scale

    return [y,E]

```

- **Annexe 9** [17] : Test du Sammon Mapping

```
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from sammon import sammon

def main():

  CIFAR=datasets.load_CIFAR10()
  X=CIFAR.data
  target = CIFAR.target
  names = CIFAR.target_names

 # Run the Sammon projection
 [y,E] = sammon(X)

 # Plot  #### ATTE SI TU AS LA FOI DE MODIFIER STP ###
 plt.scatter(y[target ==0, 0], y[target ==0, 1], s=20, c='r', marker='o',labe
 plt.scatter(y[target ==1, 0], y[target ==1, 1], s=20, c='b', marker='D',labe
 plt.scatter(y[target ==2, 0], y[target ==2, 1], s=20, c='y', marker='v',labe
 plt.title('Sammon projection of CIFAR10 data')
 plt.legend(loc=2)
 plt.show()

if __name__ == "__main__":
    main()
```

- **Annexe 10** [19] : Principe de sélection de données



- **Annexe 11**

  Algo/Pseudo-Code pour la sélection

  get indices of data.frame columns (pixels) that exist twice

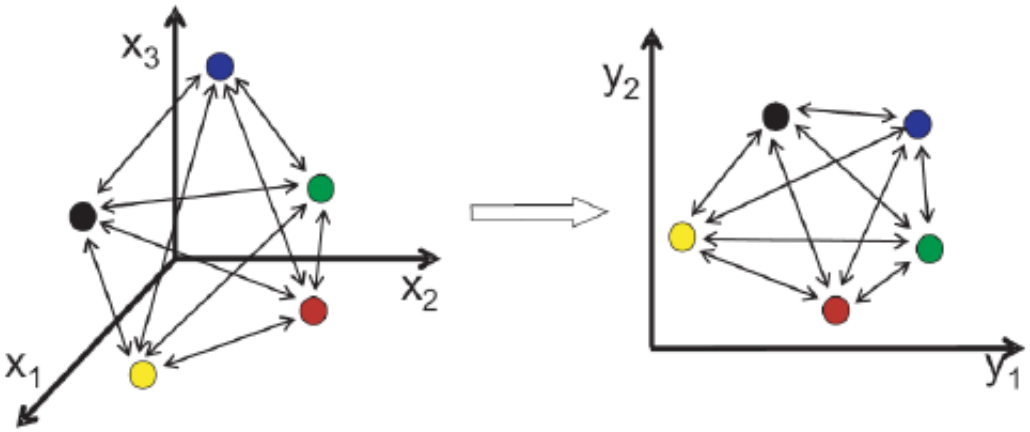
```
for i in range(data.length+1):
  for j in range(i+1, data.length+1):
    if data[j] == data[i]:
      badCols.append(j)
```

remove those "bad" columns from the training and cross-validation sets

```
train <- train[, -badCols]

cv <- cv[, -badCols
```

- **Annexe 12**

  [20] : Principale intêret de la visualisation



- **Annexe 13**

  Pseudo-Code - Moyenne

```
list = {}

Pour i allant de 1 à 256,

  Faire
    sum = 0
    Pour i allant de 1 à 1000,
      Faire
        sum += matrice[i,j]
    moyenne = sum/1000
    list <- moyenne
```

Pseudo-Code - Ecart-Type

```
list = {}

Pour i allant de 1 à 256,

  Faire
    ecart = 0
    Pour i allant de 1 à 1000,
      Faire
        ecart += (matrice[i,j] - moyenne)^2
    ecart = ecart/1000
    list <- ecart
```

Pseudo-Code - Maximum

```
list = {}

Pour i allant de 1 à 256,

  Faire
    max = 0
    Pour i allant de 1 à 1000,
      Faire
        Si matrice[i,j] > matrice[max]
        max = j
    list <- matrice[i,j]
```

Pseudo-Code - Minimum

```
list = {}

Pour i allant de 1 à 256,

  Faire
    min = 1000
    Pour i allant de 1 à 1000,
      Faire
        Si matrice[i,j] < matrice[min]
        min = j
    list <- matrice[i,j]
```