NSI Terminale Spé | Prévoir les survivants du Titanic

J. Ple J. Irdel N. Saffore

Abstract

Problématique : comment les k plus proches voisins peuvent aider à résoudre le cas du Titanic ?

1 Introduction

Ce programme permet de prédire à partir de six caractéristiques (qui sont: PClass, Sex, Age, SibSp, Parch et Fare) une cible (Survived) qui est la survie ou non du passager.

2 Aspect théorique

Le but et de permettre de prédire les données cibles manquantes, tout en ayant une précision satisfaisante.

3 Part Sociétale

L'apprentissage automatique peut aider à mieux comprendre certains éléments de notre histoire. En effet, nous avons l'exemple du *Vesuvius Challenge* par lequel un modèle d'apprentissage automatique à réussi à reprédire le contenu d'un papyrus calciné. Dans notre cas, nous pouvons utiliser l'apprentissage automatique pour connaitre la survie ou non d'un passager du Titanic

4 Méthodologie

4.1 Prétraitement des données

Pour mettre en oeuvre notre projet, nous allons d'abord nettoyer le jeu de données. C'est-à-dire utiliser des méthodes permettant de sélectionner les données utilisées et de les rendre cette traitable par notre modèle.

Pour ce faire, nous commençons d'abord par sélectionner les caractéristiques utiles à la prédiction, tel que (PClass, Sex, Age, SibSp, Parch et Fare) mais aussi la cible (Survived) en utilisant la méthode drop du module pandas.

Dans un second temps, nous adressons au problème de manque de données de certaines colonnes en calculant la moyenne entière (int).

Enfin, nous convertissons notre ensemble de données en liste afin de simplifier le traitement de l'information.

4.2 Implémentation du KNN

⇒ Nous choisisssons la ***norme Euclidienne*** qui nous sert de fonction de mesure pour notre algorithme :

$$\|\overrightarrow{AB}\| = \sqrt{\sum_{i=1}^{n}(x_{B_i} - x_{A_i})^2}$$

qui est définie comme cela en géométrie, pour deux points A correspondant à notre cible et B correspondant à un élément de notre ensemble d'entrainement à dimension n.

⇒ On peut visualiser la valeur des k plus proches voisins avec une liste de voisins en fonction de k , le premier voisin étant le voisin le plus proche de A,

$$L = [\text{voisin } 1, \text{voisin2}, \text{voisin3}, ..., \text{voisin k}]$$

On défini $\hat{y}$ comme la valeur la plus occurente de L soit le mode de L :

$$\hat{y} = \text{mode}(L)$$

4.3 Évaluation

Puis afin de trouver la meilleure valeur de k nous utilisons la ***méthode d'Elbow*** ou méthode du coude :

(exemple: si $\hat{y} = 1$ cela implique que notre modèle prédit que l'humain survie) (1)

⇒ Pour rappel avec notre colonne ["survivant"] nous avons déjà la connaissance de la survie ou non de l'humain, c'est notre valeur y true.

(prenons pour exemple qu'il est mort y true = 0) (2)

⇒ On peut calculer le coût dès à présent car nous avons nos deux variables, avec la fonction **MSE, Mean Squared Error** ou Erreur Quadratique :

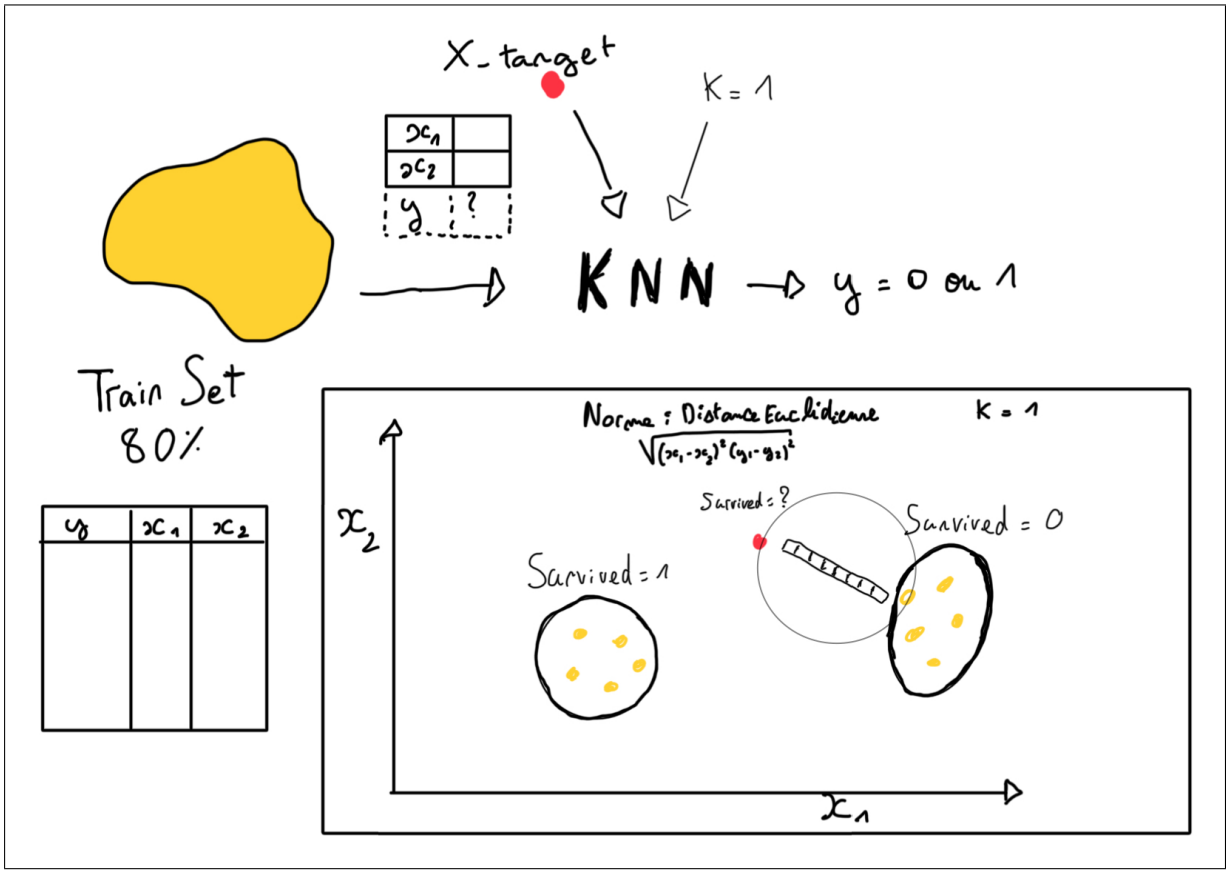
$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3}$$

(Pour notre exemple nous pouvons dire que $\text{Loss} = (0-1)^2 = 1$ ainsi si notre coût pour cette itération est de 1, notre précision est donc de 0 car la précision du modèle est calculée par accuracy $= 1 - \text{Loss}$)

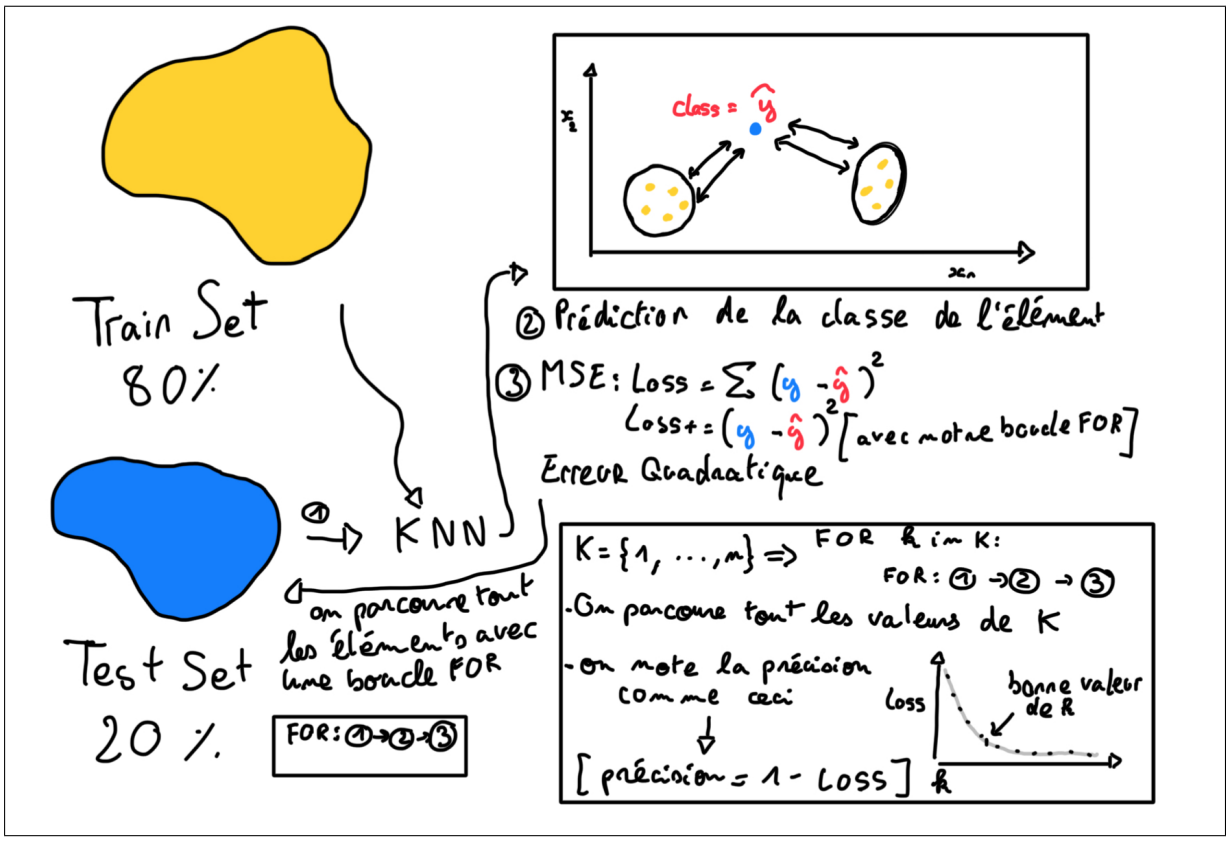
⇒ MSE nous renvoit une statistique de notre coût entre 0 et 1 et le calcul s'effectue sur la taille de notre ensemble de test n.

⇒ enfin on calcule **pour chaque k** : **la valeur du coût** en utilisant deux boucles, avec une complexité de $\mathcal{O}(n^2)$ donc !! et on affiche le graphique.

4.4 Illustrations de la méthodologie



(a) Fonctionnement de l'algorithme KNN



(b) Méthode D'Elbow, n est le nombre d'éléments de notre dataset

Figure 1: Visualisation de la méthodologie

5 Résultats et observations

$\Longrightarrow$ notre algorithme arrive à 68% de précision pour $k = 40$, nous pouvons utiliser $k = 20$ car cette valeur de k est un compromis entre efficacité et précision, selon la méthode du coude.

$\Longrightarrow$ nos résultats coïncident avec d'autres méthodes plus approximatives, par exemple il y a la méthode du pouce, par lequel k $=\sqrt{m}$, m correspondant à notre nombre d'échantillons qui est approximativement de 900 donc k $=\sqrt{900} = 30$. Nous observons qu'à partir de 30 notre coût converge :

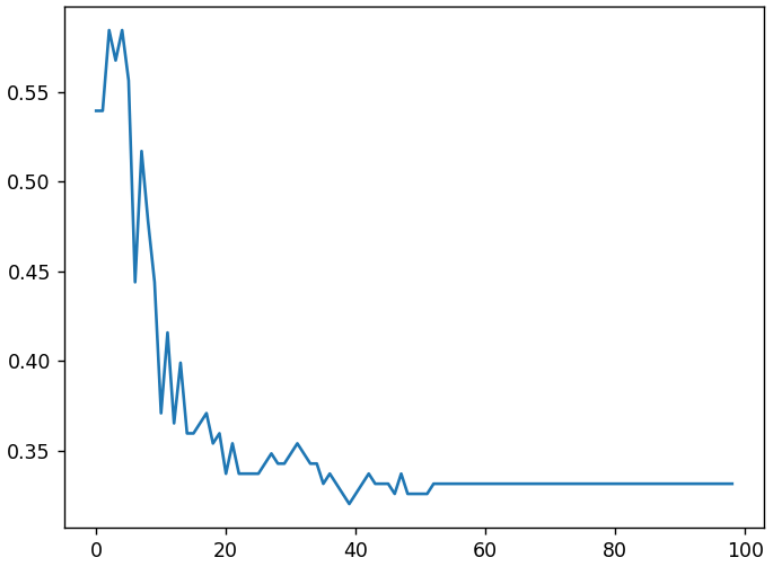


Figure 2: Courbe du coude obtenue avec $k = 100$ et la fonction de coût MSE.

6 Carnet de bord

- Jean-Baptiste : 09/09 – suppression des colonnes inutiles
- Jules : 09/09 – encodage des valeurs
- Jean-Baptiste : 14/09 – remplacement des valeurs NaN par la moyenne
- Jules : 15/09 – mélange aléatoire du dataset et division en train/test
- Jean-Baptiste : 20/09 – implémentation du KNN et méthode d'Elbow

7 Outils et ressources

7.1 Ressources

- Documentation Pandas
- Stack Exchange
- The Elbow Method: Finding the Optimal Number of Clusters
- StatQuest: K-means clustering

7.2 Outils logiciels

Python, Pandas.

7.3 Matériel

Ordinateur personnel.

7.4 Répartition

Travail réparti équitablement via Discord.

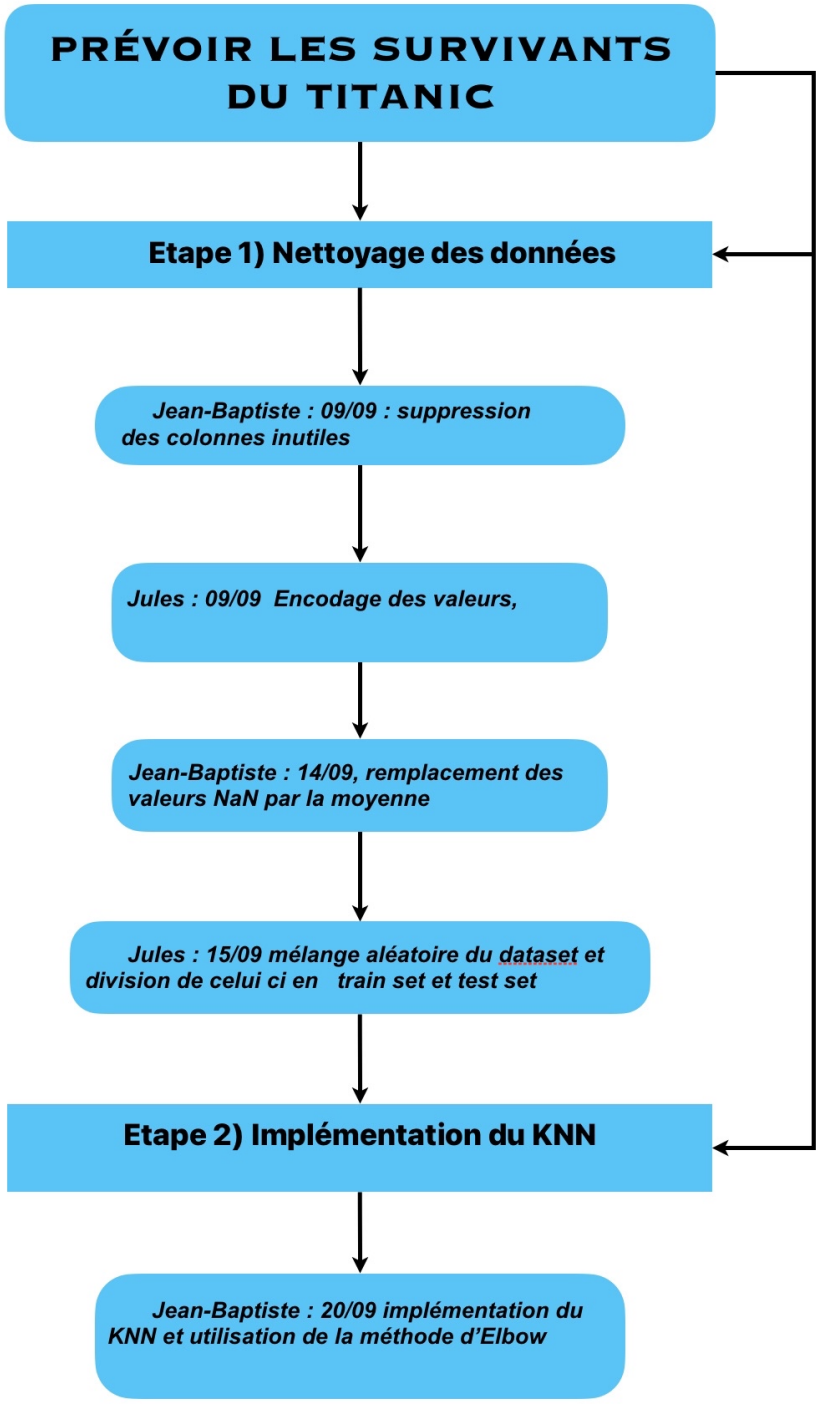


Figure 3: Carte Mentale

```
# Importation des biblioth ques
```

```
import pandas as pd
from math import *
import matplotlib.pyplot as plt
import math
import statistics
from statistics import mode
import numpy as np

# Importation des donn es
data = pd.read_csv(r"NNs from sractch\Class_Model\01_00_titanic.csv")

# Labelisation et suppression des donn es triviales comme le Ticket, qui n'appo
nom, la cabine, l'id du passager et le lieu d'embarquement
labels = data["Survived"]
data = data.drop(["Survived", "Ticket", "Cabin", "Name","PassengerId","Embarked"

#Encodage pour le sexe
data["Sex"] = data["Sex"].map({"male": 1, "female": 0})

data = data.fillna(data.mean()).astype(int) #Moyenne des donn es manquantes

#M lange des donn es
data_melange = data.sample(frac=1, random_state=42).reset_index(drop=True)
labels_melange = labels.loc[data_melange.index].reset_index(drop=True)

#R partision de 20% et 80%
test_taille = int(0.2 * len(data_melange))


test_x = data_melange[:test_taille]  #178 lignes pour faire des test
test_y = labels_melange[:test_taille]

train_x = data_melange[test_taille:] # 713 lignes or test
train_y = labels_melange[test_taille:]

# on transforme les donn es en liste
data_list = train_x.values.tolist()
data_target = train_y.values.tolist()

def distance_euclidienne(x1):
    """ fonction distance euclidienne, adapt e pour toutes les dimensions"""
    # n'ayant pas trouv  d'autres moyens, nous avons d fini cible en tant que
    global cible
    somme = 0
    for i in range(len(x1)):
        somme += (x1[i] - cible[i])**2
    distance = math.sqrt(somme)
    return distance

def knn(X,k):
    """la fonction k plus proches voisins prend en entr e :
```

```
    - X : les donn es
    - k : le nombre de voisins    prendre en compte"""
    # on utilise l   aussi une variable globale, qui est justifi   car nous avon
    global data_target
    # on calcule la norme, donc la distane entre le point cible et les autre poi
    X_sorted = sorted(X,key=distance_euclidienne)
    # on range notre liste, cela nous renvoit ce type de r sultat [(0, 102), (3
    lst = list(enumerate(X_sorted))
    # donc on ne s lectionne que le deuxi me    lment    qui nous int resse c'e
    lst.sort(key = lambda x: x[1])
    nearest_neighbors = []
    for i in range(k):
        nearest_neighbors.append(lst[i])
    nns_index = []
    # on s lectionne la ligne correspondant au voisin : nearest_neighbors[i][0]
    for i in range(len(nearest_neighbors)):
        nns_index.append(data_target[nearest_neighbors[i][0]])
    return nns_index

def mse(y_true,y_pred):
    """ on calcule l'erreur moyenne"""
    return (1/len(y_pred)) *  np.sum((y_true-y_pred)**2)

def comptage(nns_index):
    """permet de compter quelle est la classe la plus importante
    en calculant le mode : en statistique la valeur la plus fr quente"""
    return mode(nns_index)


test_x = test_x.values.tolist()
y_true = np.array(test_y.values.tolist())
y_pred = []
loss_list = []
# on parcoure toutes les valeurs de k jusqu'au nombre d'  chantillon    du test set
for k in range(1,len(test_x)):
    y_pred = []
    for i in range(len(test_x)):
        # on d fini x_cible
        cible = test_x[i]
        # on calcule notre liste L de taille k
        nns_index = knn(data_list,k)
        # on fait apparaitre chaque valeur de y hat
        y_pred.append(comptage(nns_index))
    y_pred = np.array(y_pred)
    # on calcule le co t
    loss = mse(y_true,y_pred)
    loss_list.append(loss)

# on affiche le graphique
iterations = np.arange(len(loss_list))
plt.plot(iterations,loss_list)
```

```
plt.show()
```