PONCET Clara & VELUT Lucile G1

PREDICTION CANCER DU SEIN

MODELISATION – IA

I – Identifier et définir le problème

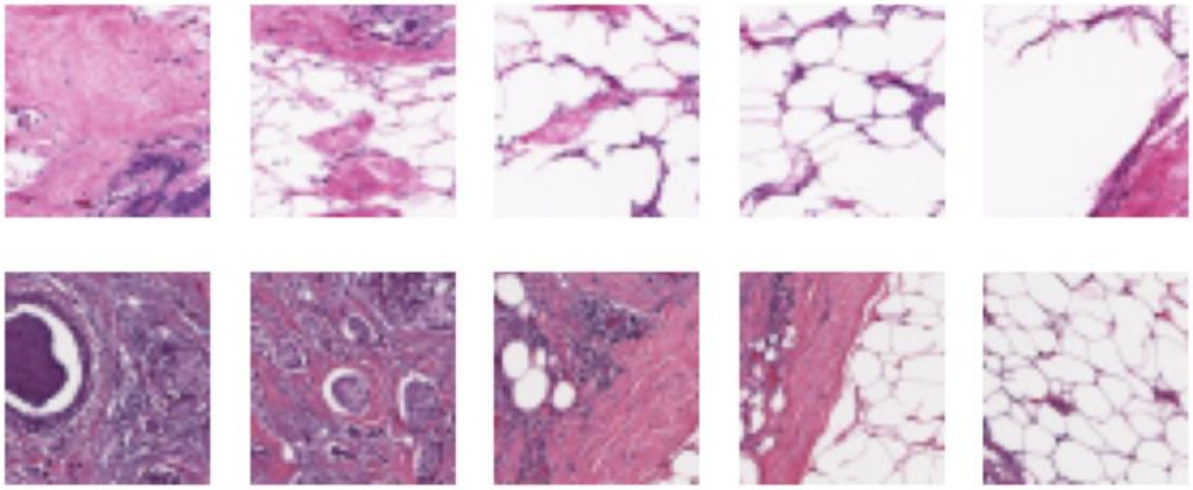
Notre projet vise à implémenter du machine learning afin de discriminer les images d'échantillons comportant des cellules cancéreuses ou non. Nous étudions ici le cancer du sein, et plus précisément les carcinomes canalaires qui sont les plus fréquents.

Pouvons-nous détecter des cellules cancéreuses sur une coupe d'échantillon de tumeur afin de localiser précisément les zones cancéreuses et ainsi évaluer la gravité du cancer ?

Pour répondre à cette question, le modèle va prendre comme entrée une image provenant de la base de données, et en sortie il dira s'il y a cancer ou non. Il s'agit donc d'une classification.

II – Comprendre les données

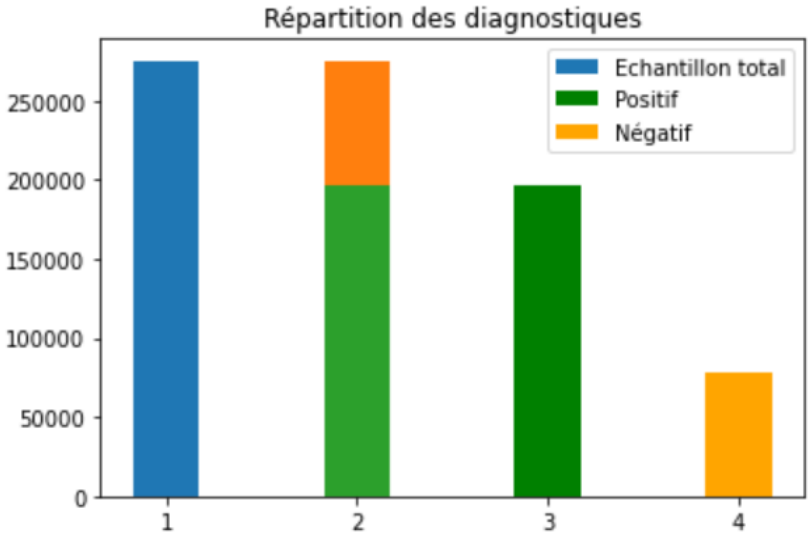
- Description des données



Affichage d'images négatives (ligne du haut) et négatives (ligne du bas)

La base de données que nous avons choisie est constituée de 277 524 photos dont 198 738 où il n'y a pas de cellules cancéreuses et 78 786 où il y en a. Cette base de données a été créé avec les photos issues de 279 patientes, pour chaque patiente il y a un dossier avec deux sous dossiers : le dossier 0 qui contient les photos non cancéreuses et le dossier 1 les photos où des cellules cancéreuses ont été identifiées. Le nom de chaque image contient le numéro de la patiente, les coordonnées de l'image ainsi que le diagnostic associé (1 pour positif, 0 pour négatif).

Pour mieux visualiser la répartition des diagnostics, nous avons tracé un diagramme en barres :



- Charger la base de données dans une structure

Pour pouvoir utiliser les données, nous les chargeons dans un dataframe. Ce dataframe est composé d'une colonne avec l'image (c'est à dire un tableau à 7200 lignes correspondant au nombre de pixels multiplié par 3, car l'image est en couleur et comporte donc trois valeurs pour chaque pixel) et d'une deuxième colonne avec le diagnostic, c'est à dire 1 si la photo présente des cellules cancéreuses et 0 sinon.

Pour cela, nous avons utilisé le code suivant :

```
#Création du dataframe

import pandas as pd

#Création de l'index correspondant au nombre d'images lues
index=range(0,275246,1)

#On "raccourcit" le tableau diagnostic :
diagnostic=diagnostic[:275246]

#Création des séries à insérer dans le dataframe
sImage=pd.Series(matImage,index=index)
sDiag=pd.Series(diagnostic,index=index)

#Création du dataframe
d = {
'image': sImage,
'diagnostic': sDiag
}

df=pd.DataFrame(d)

print(df)
df.head(5)
```

Nous obtenions ainsi le dataframe suivant :

```
                                                   image  diagnostic
0       [[0.8862745], [0.6431373], [0.80784315], [0.87...         0.0
1       [[0.85882354], [0.5882353], [0.77254903], [0.8...         0.0
2       [[0.972549], [0.9607843], [0.9764706], [0.9725...         0.0
3       [[0.9764706], [0.96862745], [0.9764706], [0.97...         0.0
4       [[0.94509804], [0.90588236], [0.92941177], [0....         0.0
...                                                   ...         ...
275241  [[0.7294118], [0.54509807], [0.6745098], [0.54...         1.0
275242  [[0.4509804], [0.27450982], [0.47058824], [0.6...         1.0
275243  [[0.50980395], [0.32941177], [0.5019608], [0.6...         1.0
275244  [[0.6901961], [0.41960785], [0.6], [0.69803923...         1.0
275245  [[0.9411765], [0.92156863], [0.93333334], [0.9...         1.0

[275246 rows x 2 columns]
```

	image	diagnostic
0	[[0.8862745], [0.6431373], [0.80784315], [0.87...	0.0
1	[[0.85882354], [0.5882353], [0.77254903], [0.8...	0.0
2	[[0.972549], [0.9607843], [0.9764706], [0.9725...	0.0
3	[[0.9764706], [0.96862745], [0.9764706], [0.97...	0.0
4	[[0.94509804], [0.90588236], [0.92941177], [0....	0.0

Par la suite, nous avons rencontré des problèmes causés par la manière dont nous avions sauvegardé les données. Nous avons donc décidé de recharger toutes les images sous une autre forme. Nous avons stocké les images dans un tableau de dimension 275246*7200. La première dimension correspond au nombre d'images sélectionnées (cf. point suivant). Dans un deuxième tableau à une seule dimension, nous avons inscrit le diagnostic des images (0 si négative, 1 si positive). Nous avons ensuite concaténé ces deux tableaux afin d'obtenir une seule structure de taille 275246*7201 afin de regrouper toutes les données dans un seul et même tableau, que nous avons sauvegardé dans un fichier npy.

```python
#On transforme notre matrice en tableau
tabImages=np.array(matImage)
tabImages=tabImages.reshape(len(matImage),7200)
print(tabImages.shape)
print(tabImages[1])

#On "raccourcit" le tableau diagnostic :
diagnostic=diagnostic[:len(matImage)]
diagnostic=diagnostic.reshape(275246,1)

#On crée un tableau qui regroupe les deux :
tabDonnees=np.concatenate((tabImages,diagnostic),axis=1)
print(tabDonnees.shape)
print(tabDonnees[1])
```

```
(275246, 7200)
[0.85882354 0.5882353  0.77254903 ... 0.9764706  0.9529412  0.98039216]
(275246, 7201)
[0.85882354 0.58823532 0.77254903 ... 0.95294118 0.98039216 0.        ]
```

- Sélectionner les photos

Lors de la création de notre fichier, nous nous sommes rendues compte que toutes les photos n'avaient pas la même taille. En effet, la plupart des photos sont de la taille 50x50 mais certaines mesurent 50x48 ou 48x50.

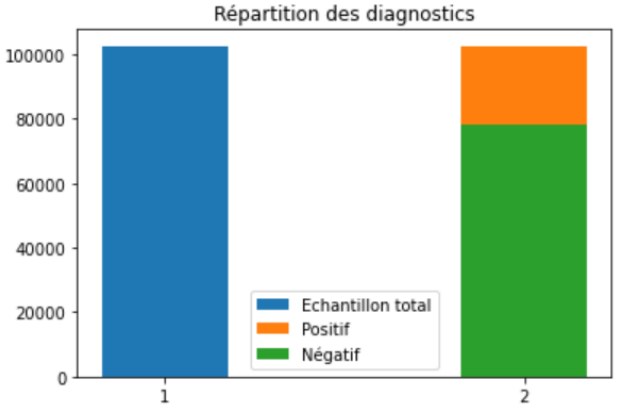
Pour résoudre ce problème, nous avons choisi de réduire toutes les images à la même taille : 50x48 (en pratique, on enlève les deux dernières colonnes à toutes les images de taille 50x50). Après avoir fait ça, nous nous sommes rendu compte qu'il restait une infime partie des images qui étaient encore plus petites, nous avons donc décidé de les ignorer et de ne pas les garder afin de ne pas perdre davantage de données.

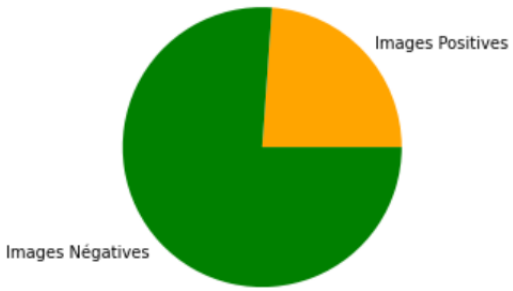
III – Choisir une méthode

Avec l'étude de cette base de données, nous cherchons à entraîner une machine à séparer les images contenant des cellules cancéreuses des images saines. Il s'agit donc d'une classification.

Nous avons donc choisi de diviser notre base en deux parties. La première partie, comprenant 75% des données, est utilisée pour « entraîner » la machine à discriminer les images. La seconde, qui contient les 25% restants de la base, est utilisée pour « tester » la machine et ainsi vérifier la précision du modèle utilisé.

Cependant, nous avons réalisé que notre base était trop importante pour être traitée par les modèles que nous avons choisis dans des temps acceptables et avec les ressources à notre disposition. Nous avons donc décidé de ne faire nos calculs avec une base de données réduite, contenant 102592 images, dont 24504 positives et 78088 négatives.





La division de la base en deux parties nous donne :

```
import sklearn as sk
from sklearn.model_selection import train_test_split


Xtrain,Xtest,ytrain,ytest = train_test_split(tabImg,tabDiag,test_size=0.25,random_state=0)
```

```
print(np.shape(Xtrain))
print(np.shape(ytrain))
print(np.shape(Xtest))
print(np.shape(ytest))
```

```
(76944, 7200)
(76944,)
(25648, 7200)
(25648,)
```

La partie « entraînement » contient 76944 images et la partie « test » en contient 25648.

IV – Les modèles

Une fois nos deux parties créées, nous pouvons lancer l'apprentissage à partir de plusieurs modèles.

Nous choisissons d'utiliser les trois modèles suivants : classification avec les plus proches voisins, classification avec un arbre de décision et classification SVM.

- Classification avec les plus proches voisins
  Il s'agit d'une classification par apprentissage supervisé. Pour chaque image, le programme regarde k voisins les plus proches (k une constante prédéfinie). Il classe ensuite l'image étudiée dans la catégorie dans laquelle se trouve la majorité de ses k voisins les plus proches.

- Classification avec arbre de décisions
  Il s'agit également d'un apprentissage supervisé. Le programme crée un arbre de décision où les nœuds sont des critères et les feuilles des valeurs discrètes. Suivant les valeurs des feuilles, l'image sera classée dans une catégorie ou dans l'autre.

- Classification SVM
  La classification Support Vector Machine est un apprentissage supervisé qui cherche à séparer les deux classes d'images par un plan. Elle consiste en la recherche du plan pour lequel l'écart entre le plan et les échantillons les plus proches, appelé marge, est le plus élevé.

V – Appliquer le modèle et commenter les résultats

- <u>Classification avec les plus proches voisins</u>

Temps d'exécution : environ 8h

Précision : 79,5%

In [10]:
```
#On commence par la méthode des plus proches voisins

KNN = KNeighborsClassifier()
clf = KNN.fit(Xtrain, ytrain)
ypredit = clf.predict(Xtest)
accuracy_score(ytest, ypredit)
```

Out[10]:
```
0.7954226450405489
```

- Classification avec arbre de décisions

Temps d'exécution : environ 20 minutes

Précision : 80,6%

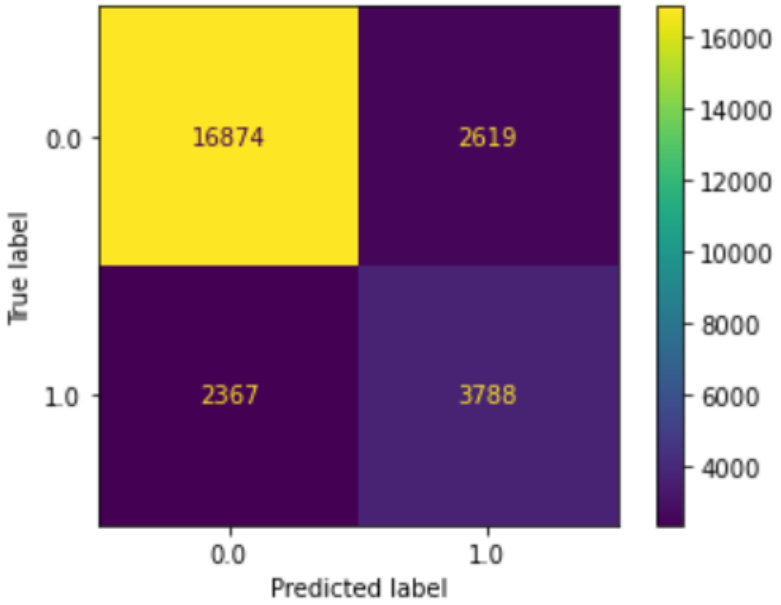
```
In [8]: #Méthode par arbre de décision

        Arbre_decision = DecisionTreeClassifier(random_state=0, max_depth=20)
        clf = Arbre_decision.fit(Xtrain, ytrain)
        ypredit = clf.predict(Xtest)
        accuracy_score(ytest, ypredit)

Out[8]: 0.8055988771054273
```

Matrice de confusion :

```
#matrice de confusion
plot_confusion_matrix(clf,Xtest,ytest)
plt.show()
```



Horizontalement, on lit les diagnostics réels des images. Verticalement, on lit les résultats donnés par la machine après sont entrainement. On peut donc voir que la machine a détecté 2619+3788=6407 images positives mais seulement 3788 d'entres elles étaient réellement négatives. La machine a effectué une mauvaise prédiction pour 2367 images qu'elle a considérées négatives alors d'elles étaient positives (case en bas à gauche).

Ce modèle est plus efficace que le précédent (sa précision est plus élevée) mais il est surtout bien plus rapide. Nous avons donc essayé de l'utiliser pour traiter la base en entière mais le tableau de données était trop grand pour être lu en une seule fois (environ 15,4Go de données).

- Classification SVM

Temps d'exécution : environ 11h

Précision : 86,4%

```
In [9]: #Méthode SVM

        clf = svm.SVC(gamma=0.001)
        clf.fit(Xtrain,ytrain)
        ypredit = clf.predict(Xtest)
        accuracy_score(ytest, ypredit)

Out[9]: 0.8637710542732376
```

Ce modèle est le plus efficace puisqu'il permet à la machine d'effectuer une bonne prédiction dans plus de 86% des cas.

Pour obtenir d'encore meilleurs résultats, nous pourrions envisager de traiter davantage de données (plusieurs paquets contenant peu d'images mis en commun par la suite).