

Rapport projet IMA205

Sommaire :

1. Introduction	2
2. Prise en main des données et des bibliothèques	2
3. Extraction des features	4
4. Le modèle	6
5. Conclusion	7

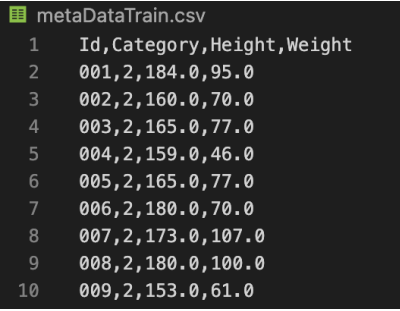
1. Introduction

Le but de ce projet est d'essayer de déterminer les pathologies cardiaques des patients (Healthy controls, Myocardial infarction, Dilated cardiomyopathy, Hypertrophic cardiomyopathy, Abnormal right ventricle) le plus précisément possible à partir d'IRM qu'ils ont effectué. Nous allons utiliser des méthodes de machine learning en python afin de classer les patients, ce qui permettra de détecter et prévenir les maladies.

2. Prise en main des données et des bibliothèques

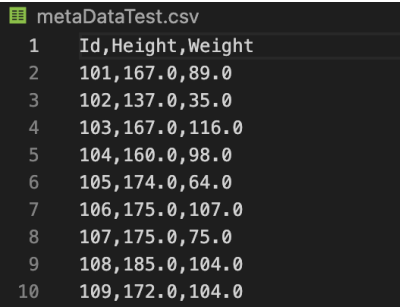
Pour ce projet nous avons deux types de données : des tableaux .csv et des images .nii.

Les tableaux csv étaient faciles à lire et à manipuler avec la bibliothèque csv de python prévue à cet effet.



```
metaDataTrain.csv
1  Id,Category,Height,Weight
2  001,2,184.0,95.0
3  002,2,160.0,70.0
4  003,2,165.0,77.0
5  004,2,159.0,46.0
6  005,2,165.0,77.0
7  006,2,180.0,70.0
8  007,2,173.0,107.0
9  008,2,180.0,100.0
10 009,2,153.0,61.0
```

Figure 1 : metaDataTrain.csv



```
metaDataTest.csv
1  Id,Height,Weight
2  101,167.0,89.0
3  102,137.0,35.0
4  103,167.0,116.0
5  104,160.0,98.0
6  105,174.0,64.0
7  106,175.0,107.0
8  107,175.0,75.0
9  108,185.0,104.0
10 109,172.0,104.0
```

Figure 2 : metaDataTest.csv

Pour les images en revanche la tâche était un peu plus compliquée. J'ai utilisé la bibliothèque torchio pour les ouvrir et pouvoir les manipuler par la suite sous la forme de tableaux numpy. Il m'a été difficile au début de comprendre que chaque fichier .nii n'était pas une simple image mais était composé de plusieurs images (correspondants à des slices, on a donc une observation sur 3 dimensions). En observant les dimensions de ces fichiers, je me suis rendu compte qu'elles n'étaient pas toutes identiques. Il fallait donc que le traitement que j'allais effectuer par la suite puisse s'adapter à n'importe quelles dimensions de l'image.

Dans un premier temps pour comprendre les données qu'on allait pouvoir manipuler par la suite j'ai affiché les images. Par exemple pour la première image du set de train en diastole :

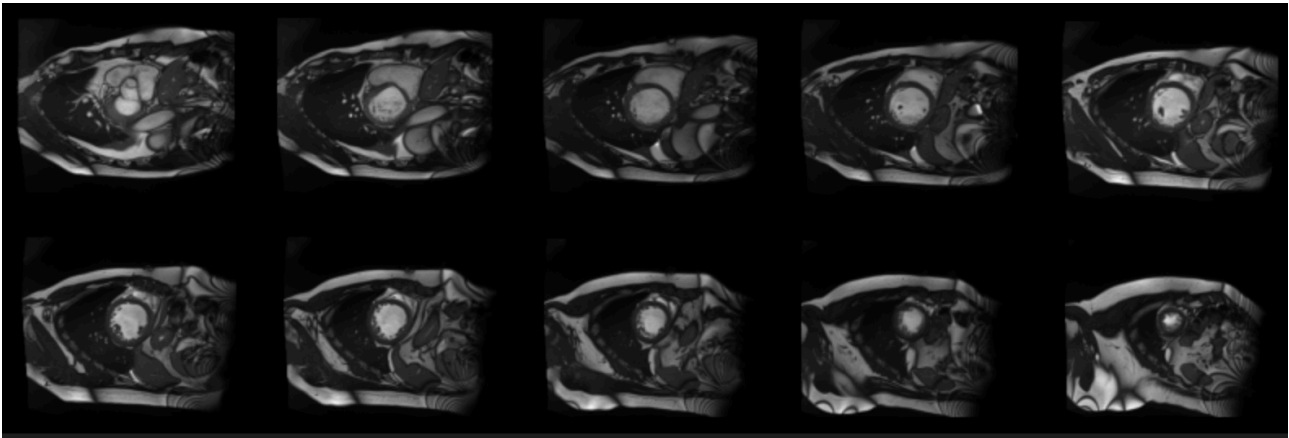
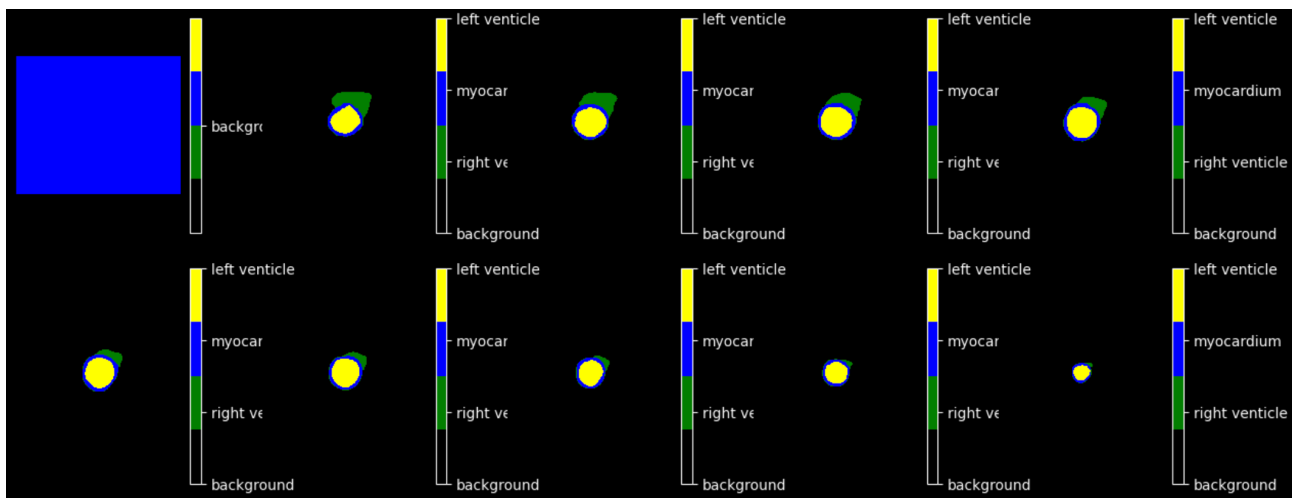


Figure 3 : ./Train/001/001_ED.nii

Les segmentations de ces images nous étaient aussi fournies (elles étaient complètes pour le train set, mais il manquait la segmentation pour le ventricule gauche dans le test set, ce que l'on verra plus tard). J'ai observé ces segmentation en colorant les labels (il ne s'agit pas d'une segmentation binaire, il y a quatre labels différents) :



- 0 - Background
- 1 - Right Ventricle Cavity
- 2 - Myocardium
- 3 - Left Ventricle Cavity

Figure 4 : ./Train/001/001_ED_seg.nii

En affichant les segmentations du test set on observe la non-segmentation du ventricule gauche :

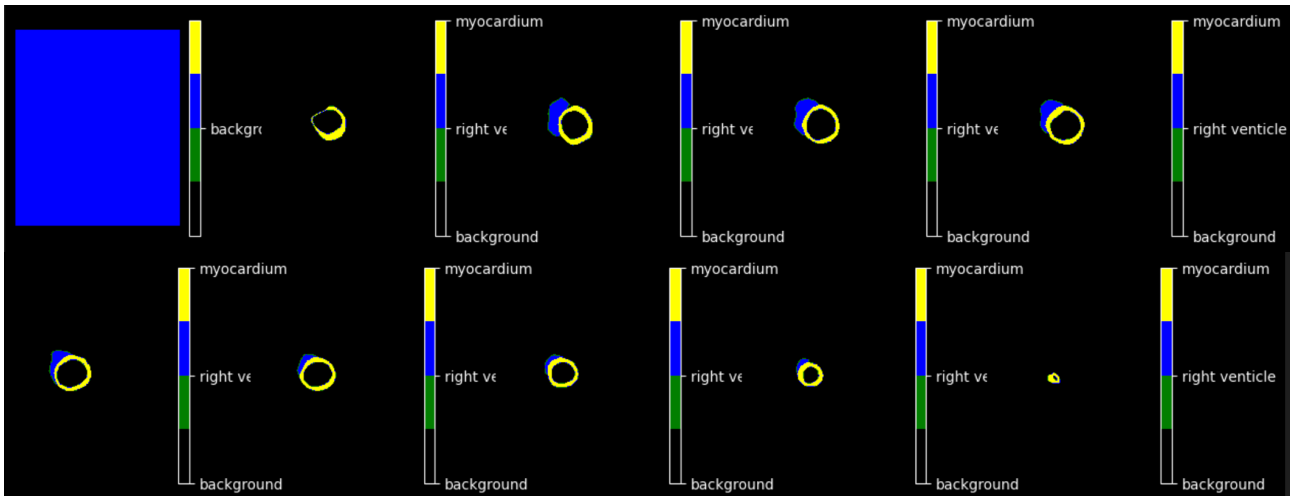


Figure 5 : ./Test/101/101_ES_seg.nii

La prochaine étape est donc de segmenter ce ventricule gauche pour pouvoir manipuler les données de segmentation du set de test. Pour cela j'ai utilisé la bibliothèque open-cv qui permet de détecter facilement les contours d'une image binaire. J'ai donc converti l'image de segmentation en image binaire, séparant les parties labellisée '0' (background et left ventricle) pour pouvoir en extraire les contours. Cv2 permet ensuite de remplir ces contours avec des fonctions pré-implémentées que j'ai utilisées. Cela marchait très bien lorsqu'il y avait effectivement des contours à traiter, mais certaines images avaient une segmentation composée uniquement de '0', auquel cas ma méthode précédente ne fonctionnait pas. Il a donc fallu que je fasse attention au cas où aucun contour n'était détecté, et m'assurer de ne pas transformer l'image (sinon, le traitement précédent va détecter les bords de l'image comme contour, et remplir toute l'image en ventricule gauche...).

La bibliothèque torchio nous permet facilement de convertir les données en tableau numpy, ce que est très pratique pour manipuler ensuite les valeurs contenues dans les images (si image est l'objet résultant de l'ouverture de l'image avec torchio, image.data.__array__() donne le tableau numpy correspondant à l'image).

3. Extraction des features

Dans un premier temps, j'ai construis y_train, les labels correspondants au set de train à partir du fichier csv fourni (les catégories de chaque patient étaient données).

Ensuite il a fallu construire les tableaux de features, que ce soit pour l'entraînement ou pour le test. En lisant les articles, j'ai compris que beaucoup de features était très intimement liées aux volumes des ventricules ou du myocarde.

Pour extraire les volumes des différents éléments pour chaque sujet, j'ai travaillé avec les images segmentées (maintenant complètes pour le train set et le test set). J'ai compté le nombre de pixels correspondant à chaque label (en sommant les pixels sur chaque slice du fichier nii). Un autre problème est survenu : tous les fichiers n'ont pas le même nombre de slices et/ou pas les mêmes dimensions en général. J'ai donc normalisé tous les volumes en les divisant par le produit des trois dimensions des fichiers nii.

J'ai ensuite rajouté aux features les métadonnées des patients (leur poids et leur taille) à partir des fichiers contenant les métadonnées.

En lisant les articles qui nous étaient donnés j'ai rajouté d'autres features, qui sont des fonctions des volumes calculés précédemment. J'ai rajouté la fraction d'éjection des ventricules (différence de leurs volumes entre la phase diastole et la phase systole), le ratio entre les ventricules droit et gauche aux phases systole et diastole et l'épaisseur du myocardium. Les fractions d'éjection et le rapport des volumes s'obtient facilement à partir des volumes précédents. Pour l'épaisseur du myocardium, j'ai émis l'hypothèse que ce dernier et le ventricule gauche était sphériques, afin de faciliter les calculs et de pouvoir calculer facilement l'épaisseur du muscle.

Pour résumer, j'ai utilisé les features suivantes (14 au total) :

- Le volume du ventricule droit en diastole
- Le volume du myocardium en diastole
- Le volume du ventricule gauche en diastole
- Le volume du ventricule droit en systole
- Le volume du myocardium en systole
- Le volume du ventricule gauche en systole
- Le poids du patient
- La taille du patient
- La fraction d'éjection du ventricule gauche
- La fraction d'éjection du ventricule droit
- Le ratio entre les ventricules droit et gauche en diastole
- Le ratio entre les ventricules droit et gauche en systole
- L'épaisseur du myocardium en diastole
- L'épaisseur du myocardium en systole

J'ai essayé de réduire le nombre de features que j'utilisais, car certaines peuvent être inutiles ou même fausser les estimations du modèle. J'ai commencé par créer une fonction qui me permettait de sélectionner manuellement les features que je gardais, afin de pouvoir faire des test pour voir si une feature en particulier pouvait augmenter ou diminuer la précision de l'algorithme.

J'ai ensuite essayé d'automatiser cela en utilisant des algorithmes pour diminuer la dimension de mes features (en sélectionner certaines). Pour cela j'ai d'abord essayé d'appliquer un PCA à mes données, en le fitant sur les données d'entraînement. Les résultats ne se sont pas améliorés (sur le score public de kaggle en tous cas), j'ai donc cherché d'autres méthodes pour sélectionner mes features.

J'ai utilisé la sélection séquentielle de feature de sklearn, qui permet de ne conserver qu'un nombre déterminé de features. Cette sélection permet de déterminer à chaque itération la meilleure feature pour le modèle (celle qui maximise la cross-validation de l'estimateur que l'on a choisi).

J'ai fais quelques essais, pour savoir combien de features donnaient un résultat optimal. J'ai obtenu mes meilleurs résultats pour 10 features sélectionnées :

- Le volume du ventricule gauche en diastole
- Le volume du ventricule droit en systole
- Le volume du myocardium en systole
- Le volume du ventricule gauche en systole
- Le poids du patient
- La fraction d'éjection du ventricule gauche
- La fraction d'éjection du ventricule droit
- Le ratio entre les ventricules droit et gauche en diastole
- Le ratio entre les ventricules droit et gauche en systole
- L'épaisseur du myocardium en diastole

4. Le modèle

Afin de classifier ces données j'ai utilisé différents types de machine learning.

J'ai d'abord essayé la méthode des KNN (K plus proches voisins) sur mes données. Je me suis rapidement aperçu qu'en utilisant toutes les features pour cette méthode on avait beaucoup d'overfitting (le modèle se spécialise trop pour le jeu de données que l'on a et n'est pas capable de s'adapter à de nouvelles données), et donc une précision plutôt faible sur les prédictions. On peut voir sur le cross-validation score sur le gridsearch :

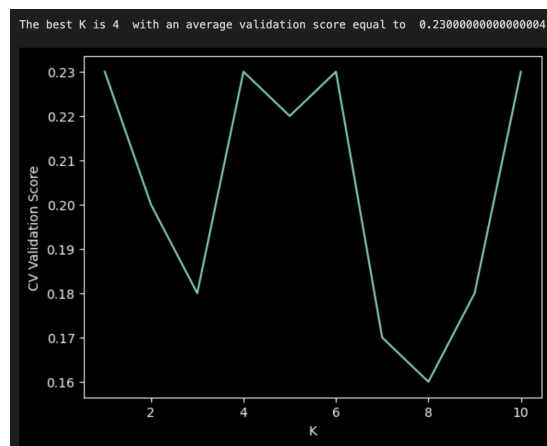


Figure 6 : Validation en fonction du nombre de voisins, avec toutes les features

En ne prenant que 6 features par exemple l'overfitting est beaucoup moins présent :

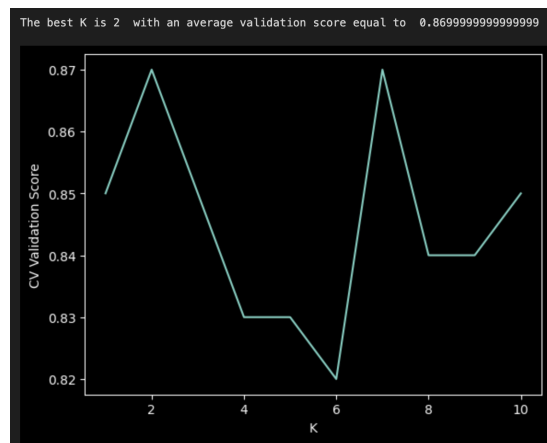


Figure 7 : Validation en fonction du nombre de voisins, avec 6 features

J'ai ensuite recherché d'autres modèles. Je me suis penché sur les arbres de décision, mais cette méthode présentait elle aussi beaucoup d'overfitting, peu importe le nombre de features sélectionnées. J'ai donc utilisé des randomForest pour résoudre ce problème. J'ai obtenu de bons résultats avec cette méthode mais j'ai essayé de l'optimiser en modifiant les paramètres par défaut. À l'aide d'un gridsearch, on trouve des paramètres différents à chaque fois à cause du caractère aléatoire des randomForest. En fixant le randomState à 1 par exemple, on peut déterminer les paramètres optimaux. Je trouve alors les paramètres suivants : {'max_features': 'sqrt', 'min_samples_leaf': 2, 'n_estimators': 80}. C'est avec ce modèle et la sélection séquentielle de feature que j'ai obtenu les meilleurs résultats.

5. Conclusion

Les méthodes décrites précédemment m'ont permis d'atteindre une précision allant jusqu'à 0.94 sur le test set en utilisant la randomForest, et le sélectionneur séquentiel de features avec 10 features (ce qui correspond à 47 images correctement classifiées sur 50). Le score de 0.94 est satisfaisant étant donné la faible quantité dont nous disposons pour entraîner notre modèle.

Lors de ce projet nous étions très limités par la quantité de données dont nous disposions. Avec plus de données il aurait été plus facile d'avoir des modèles plus précis, nous aurions pu séparer les données d'entraînement pour avoir des données de validation, afin de déterminer plus facilement la précision de la méthode que l'on emploie. Avec encore plus de données nous pourrions essayer de faire du deep learning, qui pourrait être encore plus précis. D'une manière générale augmenter le nombre de données peut régler beaucoup de problèmes (comme l'overfitting), mais l'acquisition de ces données peut être difficile, à cause du coût des IRM notamment.