

Classification des Battements de Cœur par Apprentissage Automatique

Thomas Couloud, Johan Ghrenassia, Nadir Benmounah et Julien Catanese

Résumé

Cette étude explore la classification automatique des battements de cœur à partir de signaux d'électrocardiogrammes (ECG) en utilisant des techniques de machine learning. Nous comparons les performances de classification de différents modèles sur les jeux de données Kaggle lui-même obtenu à partir des bases de données MIT-BIH et PTB. Nous comparerons les modèles de machine learning classique aux modèles d'apprentissage profond (deep learning). Nos résultats montrent que les performances varient en fonction des différents modèles et de leurs optimisations respectives. Nous avons procédé à deux types d'optimisation, celle sur les modèles et leurs paramètres et celles sur les données d'entrée du modèle. Nous mettons en évidence les limites de certains modèles et leur tendance au sur-apprentissage. Enfin nous montrons que les modèles basées sur une architecture en réseau de neurones permettent d'obtenir de meilleurs résultats de classification des battements cardiaques à partir de signaux ECG.

Table des matières

1	Introduction	2
2	Visualisation, Compréhension et manipulation des données	3
2.1	Exploration des bases de données	3
2.1.1	données brutes des bases MIT et PTB	3
2.1.2	La base de donnée Kaggle est la plus exploitable	3
2.2	Visualisation et compréhension des données Kaggle	4
2.2.1	Variables explicatives	4
2.2.2	Variable cible	5
2.3	Fusion des sets Kaggles pour obtenir notre jeu de données	5
2.4	Obtention des jeux d'entraînement, de test et de validation	6
2.5	Features engineering : la fréquence cardiaque	6
2.6	bilan de la phase de préparations des données	7
3	Modèles de Machine Learning Classique	7
3.1	Métriques pertinentes	7
3.2	Régression logistique	8
3.2.1	Régression logistique sans optimisation	8
3.2.2	Régression logistique avec optimisation	8
3.2.3	Analyse et Interprétation	8
3.2.4	Conclusion : avantages et limites du modèle	9
3.3	Arbre de décision	10
3.3.1	Arbre de décision sans optimisation	10
3.3.2	Arbre de décision avec optimisation	10
3.3.3	Analyse et Interprétation	11
3.3.4	Conclusion : avantages et limites du modèle	12
3.4	K plus proches voisins	13
3.4.1	KNN sans optimisation	13
3.4.2	KNN avec optimisation	13
3.4.3	Analyse et Interprétation	14
3.4.4	Conclusion : avantages et limites du modèle	15

4 Modèles avancés et Deep Learning	17
4.1 Gradient Boosting : CatBoost	17
4.1.1 CatBoost sans optimisation	17
4.1.2 Catboost avec optimisation	18
4.1.3 Analyse et Interprétation	18
4.1.4 Conclusion : avantages et limites du modèle	20
4.2 Réseau de neurones Artificiel (ANN, Artificial Neural Network)	21
4.2.1 ANN sans optimisation	21
4.2.2 Optimisation de l'ANN via l'architecture	21
4.2.3 Optimisation de l'ANN via les entrées	23
4.2.4 Interprétabilité et Analyse de l'ANN	23
4.2.5 Conclusion : avantage et limite du modèle	24
4.3 Réseaux de neurones récurrents	25
4.3.1 RNN sans optimisation	25
4.3.2 RNN avec optimisation	25
4.3.3 Conclusion : avantages et limites du modèle	27
5 Conclusion générale et Discussion	28
5.0.1 Comparaison de l'ensemble des modèles	28
5.0.2 Discussion et ouverture	28
5.0.3 Bilan global	29
6 Références	30

1 Introduction

Contexte et objectif global

Ce rapport a pour but de présenter le travail effectué par notre équipe lors d'un projet de détection d'anomalies dans les battements de cœur de sujets humains sains ou atteints de maladies cardiaques. Notre équipe est constituée de 4 apprenants DataScientest, et est supervisée par l'expert métier Adrien Moreau. Le challenge technique du projet sera d'apprendre à une machine à catégoriser les battements de cœur comme Normaux ou Anormaux. Pour ce faire nous avons utilisé des Électrocardiogrammes (ECG) collectés et annotés par des cardiologues. Les données sont en libre accès sur la base de donnée Kaggle (<https://www.kaggle.com/shayanfazeli/heartbeat>) qui regroupe deux collections de signaux cardiaques : MIT-BIH Arrhythmia Dataset et The PTB Diagnostic ECG Database. En se basant uniquement sur ces données ECG, nous utiliserons des outils de l'intelligence artificielle, pour apprendre à une machine à déterminer quels sont les battements de cœur sains et quels sont les battements de cœur anormaux. Pour ce faire nous testerons des modèles de machine learning classiques tel que la régression logistique, le KNN et l'arbre de décision, ainsi que des modèles plus récents tel que le CATboost. Enfin, nous entrerons dans un processus de modélisation par deep learning en implémentant des

réseaux de neurones artificiels capables de classifier des ECG. Pour finir, nous comparerons les performances de ces différents modèles, et nous tenterons de donner une explication au fonctionnement de chaque modèle.

Ce projet est avant tout à visée pédagogique, mais peut avoir une pertinence d'un point de vue scientifique et médical, car les dysfonctionnements cardiaques sont aujourd'hui parmi les plus fortes causes de mortalité au niveau mondial. Mieux comprendre et mieux détecter les anomalies dans les signaux ECG est donc crucial pour aider les professionnels de santé à diagnostiquer rapidement les problèmes cardiaques et à fournir un traitement efficace, afin d'éviter l'infarctus du myocarde par exemple. Du point de vue économique et social, la classification automatisée des battements de cœur peut réduire les coûts de soins de santé en permettant un diagnostic précoce tout en évitant les erreurs de diagnostic. Il faut noter que même un taux d'échec de 1% pour un tel système de classification automatisée appliquée potentiellement à des centaines de milliers de personnes, aurait donc un impact sur la vie de plusieurs milliers de personnes. Une recherche active sur ce sujet reste donc cruciale pour constamment augmenter le niveau de performance de l'intelligence artificielle.

2 Visualisation, Compréhension et manipulation des données

exploration des 3 bases de données Kaggle, MIT et PTB

Le premier objectif de notre projet de modélisation est de préparer un jeu de données utilisable. Cela commence par identifier une source de données ECG fiables et accessible. Puis de bien analyser leurs contenue, de bien vérifier leurs propriétés et leurs propriétés. Identifier les variables cibles et les variables explicatives. Et enfin réorganiser le jeu de données pour obtenir un set de donnée d'entraînement, un set de test, et un set de validation (qui sera isolé et conserver pour une potentielle démonstration de la généralisation de nos modèles sur des données qui n'auront jamais été "vues" par les modèles).

2.1 Exploration des bases de données

Nous avons identifier 3 bases de données ECG : PTB, MIT et Kaggle. Les deux premières bases de données contiennent les données ECG dit "brutes" , c'est à dire les tracés électrophysiologiques continues obtenues directement en sortie des électrodes d'enregistrement. Ces données sont très fiables mais peu exploitable dans leur format originale. Beaucoup d'étapes de pré-traitement serait nécessaire pour en faire des données ECG modélisable.

2.1.1 données brutes des bases MIT et PTB

Ci-dessous, une fraction de dataframe de l'ensemble de données provenant de PTB que nous avons tenté d'extraire plusieurs fichiers originaux (.dat) et que nous avons transformer en dataframe python :

Unnamed:	0	1	2	3	4	5	6	7	8 ...	120005	120006	120007	120008	120009	120010	120011	age	sex	diagnosis		
0	0	0.1755	0.1619	0.1690	0.1865	0.1970	0.2160	0.2200	0.2165	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	87.0	female	Reason for admission: Myocardial infarction	
1	0	0.0655	0.0725	0.0655	0.0600	0.0535	0.0560	0.0790	0.0950	0.1510	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	87.0	female	Reason for admission: Myocardial infarction
2	0	0.3605	0.4115	0.4200	0.4295	0.3900	0.3605	0.3385	0.3425	0.3445	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	85.0	female	Reason for admission: Myocardial infarction
3	0	-0.8430	-0.8395	-0.8405	-0.8455	-0.8515	-0.8490	-0.8515	-0.8460	0.8480	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	80.0	female	Reason for admission: Myocardial infarction
4	0	-0.0310	-0.0305	-0.0255	-0.0205	-0.0345	-0.0385	-0.0430	-0.0570	-0.0725	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	80.0	female	Reason for admission: Myocardial infarction

Figure 1-VS : Dataframe PTB

En utilisant l'ensemble des fichier a disposition nous avons put recréer un dataframe a 120015 colonnes et 549 lignes, avec un ligne par patient (par fichier .dat). La première colonne est l'identifiant du patient. Les 3 dernières colonnes donnent accès respectivement à l'âge, au sexe et au diagnostic du patient (récupérer sur des fichiers .hea). Toutes les autres 120011 colonnes correspondent aux points d'enregistrement continue dans le temps et sont associés a une valeur en mV.

Cependant comme les temps d'enregistrements de chaque patients sont différents, la longueur de chaque varie et certaine ligne comporte donc des NAN. Ces données nécessitent donc d'être traitées.

Nous avons opéré de la même manière pour explorer la base de données du MIT :

Unnamed:	0	1	2	3	4	5	6	7	8 ...	649993	649994	649995	649996	649997	649998	649999	age	sex	drug	
0	0	0.260	0.260	0.260	0.260	0.260	0.260	0.260	0.260	...	-0.065	-0.125	-0.220	-0.305	-0.370	-0.380	0.00	73	F	Digoxin, Nitroglycerine, Propranolol
1	0	-0.065	-0.065	-0.065	-0.065	-0.065	-0.065	-0.065	-0.065	...	-0.370	-0.510	-0.490	-0.410	-0.365	-0.325	0.00	59	M	Albuterol, Isoproterenol
2	0	0.210	0.210	0.210	0.210	0.210	0.210	0.210	0.210	...	0.025	0.015	0.000	0.015	0.015	0.025	0.00	76	F	Albuterol, Isoproterenol
3	0	-0.470	-0.470	-0.470	-0.470	-0.470	-0.470	-0.470	-0.470	...	-2.165	-2.085	-1.965	-1.825	-1.740	-1.28	51	F	Propranolol	
4	0	-0.015	-0.015	-0.015	-0.015	-0.015	-0.015	-0.015	-0.015	...	-1.210	-1.470	-1.750	-1.960	-2.160	-2.55	69	M	Digoxin, Norpace	
5	0	0.115	0.115	0.135	0.135	0.135	0.135	0.135	0.135	...	0.000	0.005	0.005	0.010	0.000	0.000	0.00	89	M	None
6	0	0.115	0.115	0.135	0.135	0.135	0.135	0.135	0.135	...	0.085	0.085	0.070	0.070	0.075	0.080	0.00	56	F	None
7	0	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.035	...	-0.370	-0.550	-0.745	-0.790	-0.795	-1.28	47	F	Digoxin, Lasix	
8	0	0.050	0.050	0.050	0.050	0.050	0.050	0.050	0.050	...	0.070	0.000	0.100	0.095	0.080	0.075	0.00	83	M	Hydrochlorothiazide, Lasix

Figure 2-VS : Dataframe MIT

Ce dataframe ci est composé de plus de 650 000 colonnes 48 lignes. Les données de type catégorielles sont ici âge, sexe et médicaments prescrit (3 dernières colonnes). Ainsi, nous avons actuellement 2 dataframes de dimensions différentes ne comportant pas les mêmes informations sur les patients. Les signaux n'ont pas la même durée non plus, ni les même fréquences d'échantillonnages. Par conséquent, l'extraction des signaux ECG de ces dataframes pour une éventuelle fusion des données dans un formats compatible nécessiterait un travail énormes de pre-processing et d'organisation des données pour être exploitable pour nos modèles.

2.1.2 La base de donnée Kaggle est la plus exploitable

La base de données Kaggle contient des données issues des deux autres bases PTB et MIT décrites précédemment. Le gros avantage de Kaggle est que les données ont déjà subit les phases de pré-traitement nécessaire a l'utilisation des données pour la modélisation.

PTB NORMAL

0	1	2	3	4	5	6	7	8	9 ...	178	179	180	181	182	183	184	185	186	187
0	1.000000	0.900324	0.358695	0.051459	0.046596	0.126823	0.133304	0.119125	0.110616	0.113047	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.000000	0.794581	0.375370	0.116883	0.000000	0.171923	0.238359	0.293754	0.345680	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.000000	0.791482	0.423196	0.180712	0.000000	0.007186	0.060302	0.077002	0.074957	0.077342	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.000000	0.478893	0.056703	0.064176	0.081289	0.073723	0.065619	0.048774	0.054478	0.041643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.000000	0.867238	0.201360	0.099349	0.141336	0.120534	0.080516	0.059639	0.059346	0.010020	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4046 rows x 188 columns

PTB ABNORMAL

0	1	2	3	4	5	6	7	8	9 ...	178	179	180	181	182	183	184	185	186	187
0	0.932230	0.895679	0.898196	0.529626	0.308775	0.051397	0.010443	0.074973	0.087229	0.635100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.000000	0.608941	0.384181	0.254237	0.223567	0.278638	0.253430	0.184482	0.153349	0.121072	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.000000	0.951613	0.923963	0.853303	0.791659	0.734255	0.672643	0.685100	0.676007	0.674735	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.977819	0.895261	0.290129	0.023248	0.142229	0.223960	0.328095	0.387837	0.391701	0.389094	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.935618	0.805161	0.060518	0.000000	0.022741	0.480789	0.454829	0.319853	0.266874	0.208411	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10506 rows x 188 columns

MIT TRAIN

	0	1	2	3	4	5	6	7	8	9	...	178	179	180	181	182	183	184	185	186	187
0	0.977941	0.925471	0.681373	0.245996	0.154412	0.191176	0.151961	0.085784	0.058824	0.049620	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.960114	0.862348	0.461538	0.196581	0.094017	0.125356	0.099715	0.08319	0.074074	0.082621	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	1.000000	0.653493	0.186486	0.097070	0.070270	0.065945	0.065757	0.043243	0.040564	0.045946	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.925114	0.657476	0.541436	0.276243	0.196133	0.077348	0.071923	0.060773	0.06298	0.058011	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.967136	1.000000	0.830986	0.586854	0.356808	0.248826	0.145540	0.089202	0.117371	0.150235	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

87554 rows × 188 columns

MIT TEST

	0	1	2	3	4	5	6	7	8	9	...	178	179	180	181	182	183	184	185	186	187
0	1.000000	0.752624	0.111570	0.000000	0.080579	0.078516	0.066116	0.049587	0.047521	0.035124	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.908425	0.783883	0.531136	0.362637	0.366300	0.344322	0.333333	0.307692	0.296703	0.300066	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.730000	0.212389	0.000000	0.179469	0.101770	0.101770	0.110619	0.123894	0.115044	0.132743	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.900000	0.910447	0.681350	0.472917	0.229167	0.068750	0.000000	0.004167	0.014583	0.054167	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.570470	0.399329	0.238255	0.147651	0.000000	0.003356	0.040266	0.086537	0.070470	0.090004	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

21892 rows × 188 columns

Figure 3-VS : Le set de Kaggle se décompose en deux dossiers comportant chacun deux datasets. Un dossier PTB avec un dataset contenant les signaux anormaux et un autre les normaux et un autre dossier MIT avec un dataset Train et Test tout deux composés de 5 types de signaux (un pour un ECG normal et 4 autres pour des anomalies).

Comme le montre les tableaux de la figure 2-VS, chacun de ces dataframes sont tous composés du même nombre de colonnes (188) avec les labels de la variable cible contenus dans la dernière colonne. De plus ces dataframes ont été homogénéisés à la même fréquence d'échantillonage (125Hz) ce qui fait que tout les échantillons sont de durée de 1,5sec. Aucun ne comporte de valeurs manquantes (pas de NAN) et les signaux trop courts sont complétés par des 0 de sortes que chaque signal fasse le même nombre de points ("zeros padding").Les valeurs en mV ont été normalisées par une fonction minimax (min=0 et max=1). La variable explicative est donc composée d'une série de 187 points formant un pattern d'ondes électriques représentant un cycle de battement de coeur au cours du temps qu'on appellera ici "signal ECG" ou "battement ECG". Les détails de la phase de pre-processing des données Kaggle sont décrits dans l'article de Kachuee et al., 2018 (ref[1]).

Suite à ce début d'exploration, la base de données Kaggle nous semble la mieux adaptée pour le projet car les données sont propres et organisés de manière propice à la modélisation.

2.2 Visualisation et compréhension des données Kaggle

2.2.1 Variables explicatives

Commençons par visualiser nos variables explicatives, autrement dits les points ECG successifs formant un cycle de battement cardiaque.

On note que les signaux forment des patterns distincts et sont difficiles à comprendre à première vue. Nous nous aiderons d'un schéma explicatif pour mieux comprendre les signaux (figure 5-VS)

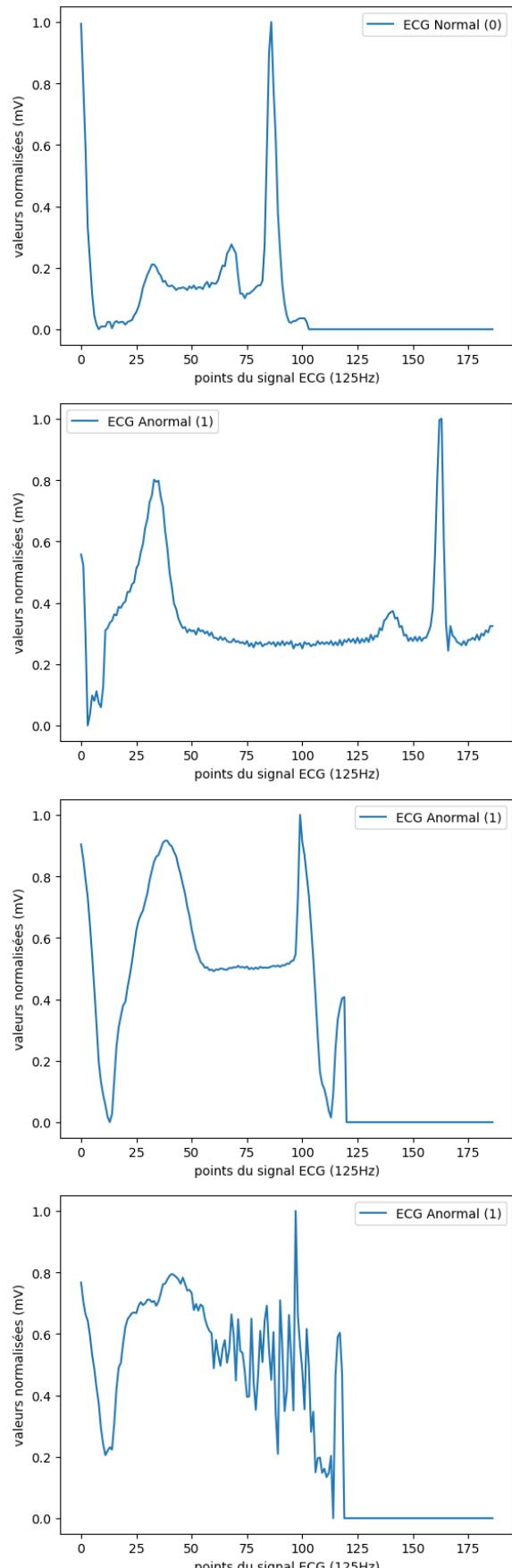


Figure 4-VS : Exemples de battements ECG. Les plots sont obtenus avec la librairie matplotlib

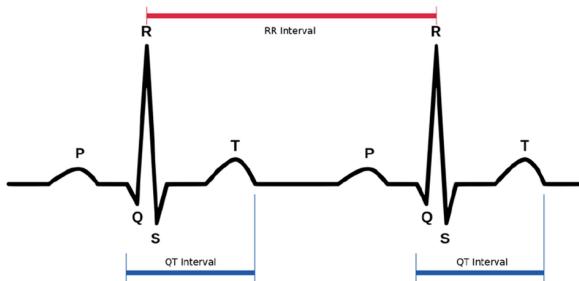


Figure 5-VS : ECG type

Sur ce dessin nous pouvons comprendre qu'au cours d'un cycle (un battement) différentes ondes P, Q, R, S, T se succèdent sur l'enregistrement. Ces ondes représentent le passage d'un signal électrique naturelle à travers le cœur correspondant à une série ordonnée de contractions/relaxation musculaire au sein des différentes parties du cœur. Ce cycle normal peut-être altéré pour certaines pathologies.

Le signal normal vu plus haut (figure 4-VS) correspond bien au dessin descriptif (figure 5-VS).

Par contre, on remarque que le signal anormal diffère au moins en certains points de l'ECG. Il existe 5 grand type de pathologies cardiaques associé à des patterns d'ECG connus des cardiologues voir la référence suivante pour plus de détails^[1].

2.2.2 Variable cible

Notre objectif est de faire une classification binaire entre deux classe de signaux. Notre variable cible doit donc être un label binaire, qui indique si la forme du signal ECG est Normal (= 0) ou Anormal (= 1). Dans notre cas, les données ont déjà été labélisées par des cardiologues (dernière colonne de chaque dataframe). Nous allons donc extraire cette information des dataframe de Kaggle pour obtenir un vecteur de label.

Puis nous voulons visualiser la répartition des classes au sein de chaque dataframe (figure 6-VS).

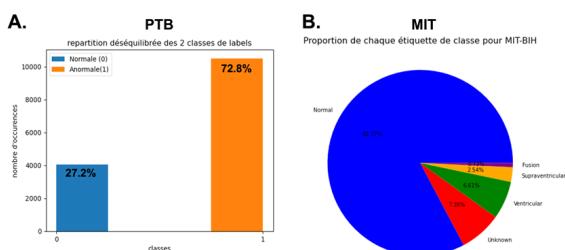


Figure 6-VS : répartition des classes. A. pour PTB database : 2 classes d'ECG déséquilibrés ; Ces données sont issues de 294 patients malades atteints de 9 maladies cardiaques différentes B. pour MIT database : 5 classes d'ECG libellés déséquilibrés ; Ces données sont issues de 47 sujets choisis pour leurs types d'arythmies rares

Le premier constat important est que les données sont déséquilibrées. Le dataframe PTB contient plus de signaux ECG Anormaux (72%) alors que le dataframe MIT contient au contraire plus de signaux ECG Normaux(83%). La fusion pourrait donc suffire à rééquilibrer, nous verrons que cela n'est pas le cas par la suite (figure 7-VS).

Le deuxième constat important est que la base de données MIT contient 5 classes. Ceci est du au fait qu'il existe plusieurs type d'anomalies cardiaques. Il va donc nous falloir les regrouper en une seule classe d'ECG Anormale, afin d'obtenir deux classes comme pour la database PTB.

2.3 Fusion des sets Kaggles pour obtenir notre jeu de données

Notre but est de fusionner deux dataframes MIT and PTB pour n'obtenir qu'un seul dataframe avec encore plus de données propre et prête à la modélisation. Nous avons commencer par homogénéisé les labels des variables cibles, en regroupant les labels du MIT des différents type d'anomalie cardiaques (1,2,3,4) en un seul label de valeur 1 (ECG Anormaux). Nous avons vu précédemment que ces dataframes sont compatibles entre eux. Nous décidons donc de les fusionner.

A présent, nous allons recalculer la nouvelle répartition des classes dans nos données fusionnées (figure 7-VS)



Figure 7-VS : Répartition des classes. On obtient plus de 70% d'ECG normaux (classe0), contre 30% d'ECG Anormaux(classe1)

Ce graphique nous apporte deux informations. D'abord, la classe normale (0) est majoritaire sur la classe anormale. En effet, il y a environ 3 fois plus de signaux normaux. Cette observation nous suggère qu'un rééquilibrage des classes pourra être utilisé lors de l'optimisation des modèles. Des procédés d'under ou over sampling ou encore smote semblent intéressants pour contrer cet éventuel biais.

2.4 Obtention des jeux d'entraînement, de test et de validation

A ce stade, nos données peuvent être divisées en plusieurs jeux, nécessaire aux étapes de modélisation. Nous créons donc un jeu de validation (10.0% des données total soit 12400 échantillons), un jeu de test (18% des données total soit, 22320 échantillons) et un jeu d'entraînement (72% des données total, soit 89278 échantillons)

Maintenant nous allons tenter d'observer si nos deux classes présentes des signaux avec des différences systématiques qui serait donc visible immédiatement via une visualisation des tendance moyenne. Nous avons choisi ici la médiane à la place de la moyenne afin d'éviter de potentiel biais du aux valeurs aberrantes. Le graphique ci-dessous décrit la médiane des différents signaux :

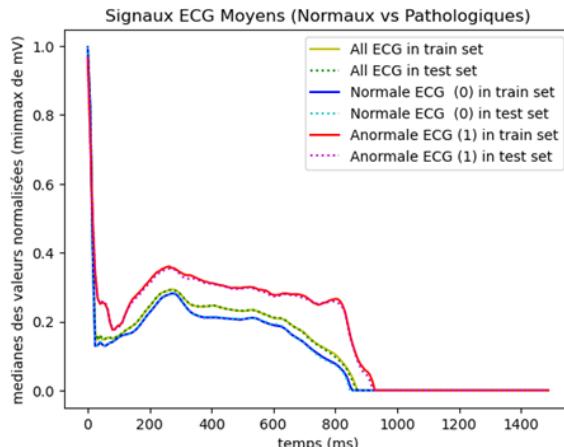


Figure 8-VS : Signaux médians en fonctions des classes.

On peut voir ici plusieurs choses. D'abord, la médiane des signaux anormaux et normaux semble suivre une forme similaire mais avec des amplitudes différentes. Les signaux commencent à diverger dès la fin du premier pic R avec une amplitude plus forte pour les ECG Anormaux. De plus le fait d'utiliser une médiane (ou moyenne) tend à supprimer les détails par exemple on distingue l'onde T à 300ms mais après cela tous semblent lissés. On perd donc la résolution temporelle fine lorsqu'on travail avec des ensembles de courbes. Nous verrons si cela peut affecter les modèles. Notons par ailleurs, que les jeux de test et d'entraînement obtiennent des valeurs médianes identiques ce qui nous indique que nos données ont été divisor de manière homogènes.

2.5 Features engineering : la fréquence cardiaque

Après l'exploration de nos données, nous avons décidé d'en extraire le temps des pics R-R afin d'en déduire par le calcul la valeur de fréquence cardiaque correspondante à chaque battement ECG. On obtient donc une nouvelle colonne dans notre dataframe, correspondant à une nouvelle variable explicative de type quantitative nommée "fcard" issue de notre feature engineering. Ces nouvelles variables pourraient venir renforcer nos modèles pour leurs prédictions en donnant l'accès à des informations supplémentaires. Vérifions la distribution des fréquences cardiaques ainsi obtenues (figure 9-VS).

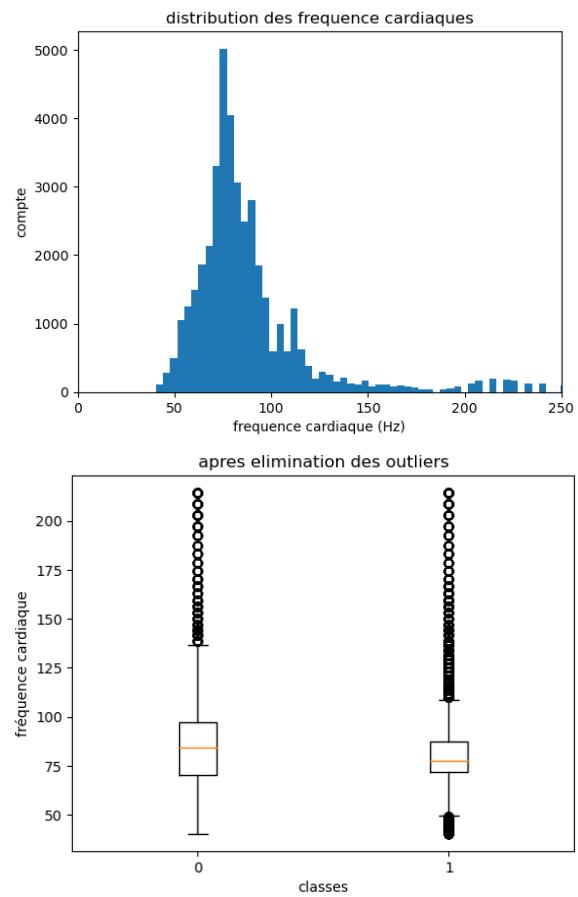


Figure 9-VS : Distribution des fréquences cardiaques

Le premier graphique décrit la distribution globale des fréquences cardiaques. La majorité des valeurs situées entre 40bpm (battements par minute) et 150bpm ce qui correspond à des fréquences attendues pour le fonctionnement normal du cœur d'humains adultes. La grande variabilité interindividuel est attendu. Nous obtenons aussi des valeurs supérieures à 220bpm ce qui est impossible physiologiquement. Ces valeurs sont aberrantes. Ceci est en grande partie due

au fait que notre algorithme de calcul basé sur les fonctions max et argmax de python, ce qui le rend très sensible à la qualité des pics R. Certains ECG sont bruités et de diverses formes (figure 4-VS) ce qui entraîne des erreurs d'estimation de fcard. Pour simplifier, nous décidons de définir un maximum à 220bpm.

Enfin, le boxplot ci-dessus (figure 9-VS) nous montre que les médianes sont proches mais que les ECG anormaux semble associé à des fréquences cardiaques plus lente, avec un temps plus long entre les pics R (ce qui ressort aussi au niveau de la figure 8-VS). De plus, la répartition (quartiles) des fréquences cardiaques anormales est plus restreinte que celle des normales. Ce qui semble indiquer moins de "flexibilité" cardiaque. Notre variable fcard est prête pour la

modélisation, nous essayerons de l'utiliser au cours de l'optimisation des modèles (voir partie ANN).

2.6 bilan de la phase de préparations des données

Avant d'aborder la phase de modélisation faisons un point rapide sur notre dataset, nous avons : - 187 colonnes composé des points d'enregistrements de l'ECG, nos variables explicatives - La colonne « class » comportant notre variable cible, ECG normal (0) ou anormal (1) - Nos classes sont déséquilibrées il faudra donc envisager des modifications - Aucune valeur nulle ou NaNs.

Notre dataset est donc prêt pour y entraîner et tester nos modèles prédictifs.

3 Modèles de Machine Learning Classique

Régression logistique - Arbre de décision - K plus proches voisins

Ici nous entrons dans la phase de modélisation du projet. C'est une étape complexe avec de nombreux choix à faire et qui nécessite donc une analyse approfondie. Elle constitue le cœur fondamental du travail d'un Data Scientist. Nous avons décidé de commencer par des modèles de classification les plus simples à interpréter afin d'obtenir une idée globale des scores de base (Baseline) auquel comparer nos modèles plus complexes.

3.1 Métriques pertinentes

L'accuracy est la proportion de prédiction correct (Vrai Positif "VP" et Vrai Négatif "VN") parmi le nombre total de cas examinés (incluant les Faux Positif FP et Négatif FN). L'accuracy ce calcul donc comme suit $(VP+VN)/(VP+VN+FP+FN)$. Nous devons faire attention aux données déséquilibrées comme c'est le cas ici pour lequel l'accuracy ne donne pas une mesure fiable. Pour cette raison ici nous l'utiliserons seulement pour évaluer le sur-apprentissage en comparant les données d'entraînement et de test au cours des itérations d'entraînement des modèles. Pour quantifier le score global nous préférerons utiliser le F1.

La précision : La précision est également appelée Positive Predictive Value. Elle correspond au taux de prédictions correctes parmi les prédictions positives, elle se calcule ainsi : $VP/(VP+FP)$. Elle mesure la capacité du modèle à ne pas faire d'erreur lors d'une prédiction positive.

Le rappel : cette métrique permet de savoir à quelle proportion le modèle détecte des signaux comme étant dans la classe 0 ou 1 et se calcule comme cela : $VP/(VP+FN)$. Cette mesure permet de vérifier que d'éventuels biais n'influencent pas les scores comme un déséquilibre des données par exemple.

Le f1-score : ce dernier est défini comme étant la moyenne harmonique de la précision et du rappel. Cette mesure regroupe donc les deux précédentes. Plus cette valeur est élevée, plus le modèle sera performant. Nous l'utiliserons pour évaluer le score global des modèles car permet de prendre en compte le rappel (et n'est donc pas sujette au limitation de l'accuracy).

L'importance : c'est une valeur attribuée à chaque variable explicative, qui permettra de quantifier l'influence ou l'impact de la variable dans le résultat de la classification. Par exemple, dans le cadre d'un arbre de décision, elle est définie comme le décroissement total du critère d'impureté entre un nœud et les deux nœuds suivants.

L'étude groupée de ces métriques est indispensable afin d'éviter des biais au sein du modèle qui pourrait survenir notamment dû au déséquilibre des classes.

3.2 Régression logistique

La régression logistique est largement utilisée car elle est extrêmement efficace et ne requiert pas d'énormes quantités de ressources informatiques. Elle peut être interprétée facilement et ne nécessite pas de mise à l'échelle des caractéristiques d'entrée. Ce modèle est également facile à mettre en œuvre, ce qui en fait une excellente référence pour aider à mesurer les performances d'autres algorithmes complexes. En revanche, la régression logistique n'est pas l'algorithme le plus puissant disponible. Il existe plusieurs alternatives qui peuvent créer des prédictions bien meilleures et plus complexes.

La régression logistique est un modèle statistique qui est souvent utilisé pour la classification binaire et nous l'appliquons pour classer les données en deux classes : normale et anormale.

3.2.1 Régression logistique sans optimisation

Dans un premier temps, nous avons créé un modèle de régression logistique brut sans effectuer d'optimisation ou réglage de paramètres. Ce modèle brut nous permet d'obtenir des résultats initiaux qui nous donnent une idée de la performance du modèle avant toute optimisation. Nous avons ensuite évalué la performance du modèle brut en utilisant la métrique f1-score, qui mesure l'harmonique entre la précision et le rappel.

	Classes	Précision	Rappel	F1-score
Modèle Logit	0	0,85	0,97	0,9
	1	0,8	0,45	0,57

Figure 1-LR : Tableaux des scores sans optimisations

Le modèle brut a obtenu le meilleur f1-score pour la classe 0, en raison du déséquilibre de nos classes, il y a seulement 57% des données considérées comme anormales (1). Le modèle a obtenu une précision de 80% pour la classe 1, mais son rappel n'était que de 45%. En d'autres termes, le modèle a identifié correctement 45% des données anormales, ce qui est un inconvénient majeur. Il est important de noter que la précision mesure la proportion de vrais positifs parmi les exemples prédits positifs, tandis que le rappel mesure la proportion de vrais positifs parmi tous les exemples positifs réels. Dans le cas présent, le modèle a tendance à manquer d'un grand nombre d'exemples positifs réels (classe anormale), ce qui affecte négativement sa capacité à identifier les anomalies.

3.2.2 Régression logistique avec optimisation

Nous avons ensuite cherché à corriger le déséquilibre de nos classes en utilisant une méthode d'undersampling pour une première optimisation du modèle de régression logistique. Cette méthode a permis de

rééquilibrer le rappel entre nos classes, avec un gain de 23% sur la classe anormale pour une perte de 17% sur la classe normale. Bien que cette amélioration soit satisfaisante, elle s'est traduite par une baisse significative de la précision, qui est désormais aussi faible que 52%.

Afin d'améliorer la précision tout en conservant un rappel élevé, nous avons utilisé une technique de recherche d'hyperparamètres appelée GridSearchCV. Cela nous a permis d'obtenir de trouver les meilleurs hyperparamètres pour notre modèle de régression logistique.

#	Optimisation testée	Impact sur la précision	Impact sur le rappel	Impact sur le f1-score
1	RandomUndersampling du jeu de données	0,89	0,8	0,84
	"C=1000" et "solver=lbfgs"	0,52	0,68	0,59
2	Ajout des meilleurs paramètres avec GridSearchCV	0,89	0,8	0,84

Figure 2-LR : Tableaux des scores avec optimisations

Cependant, malgré l'ajout des paramètres C et solver, cette technique n'a pas permis de corriger le manque de précision du modèle. Bien que nous ayons observé une amélioration d'un point sur le rappel de la classe anormale, les scores obtenus sur cette classe restent trop faibles voire aléatoires.

En fin de compte, le modèle de régression logistique ne fournit pas des résultats satisfaisants pour notre projet de détection d'anomalie cardiaque, car les scores sur la classe anormale ne sont pas acceptables. Nous ne pouvons donc pas retenir ce modèle pour la suite de notre développement et une analyse plus approfondie n'est pas nécessaire.

3.2.3 Analyse et Interprétation

Nous allons maintenant essayer de voir de quelle manière cette régression utilise nos données afin de mieux comprendre son fonctionnement.

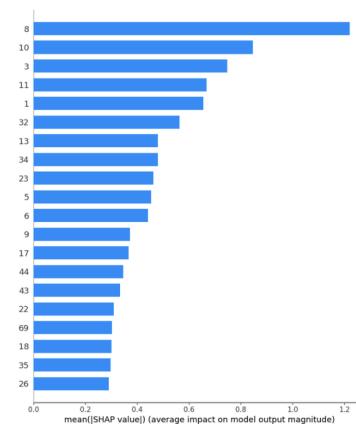


Figure 3-LR : Importance des variables

Globalement, ce graphique nous montre que les premiers points du signal ont un impact plus important sur le modèle que les points situés à la fin. En effet, les points 8, 10, 3, 11 et 1 sont dans cet ordre ceux qui influent le plus. À noter que plus ces valeurs sont grandes, plus leurs impacts sont importants.

Le graphique ci-dessous, nous montre que de grandes valeurs situées au début cet induisent la présence ou non d'un pic. Cette caractéristiques étant primordiale dans la différenciation normale ou anormale d'un ECG, on peut comprendre que ces valeurs aient une importance conséquente pour le choix du modèle. Pour vérifier davantage cette hypothèse, nous pouvons placer les points importants sur un ECG et regarder si cela correspond à des zones importantes du signal.

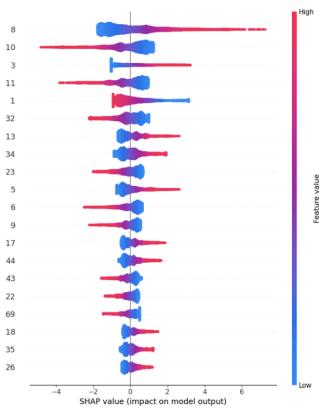


Figure 4 - LR : Importance des variables

Prenons à présent un signal mais en plaçant les points important détectés par Shap :

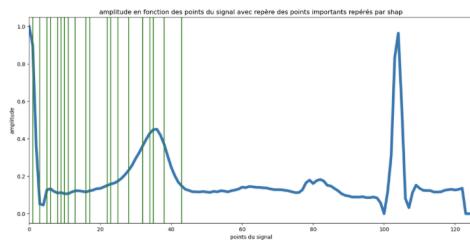


Figure 5 - LR : Valeurs importantes placées sur un ECG de classe 0

On constate sur cet exemple que le modèle cherche à discriminer les classes par les différences entre les caractéristiques des signaux normaux et anormaux (présence ou absence de pic par exemple). Ci-dessous d'autres exemples de signaux pris au hasard nous permettent d'appuyer cette hypothèse.

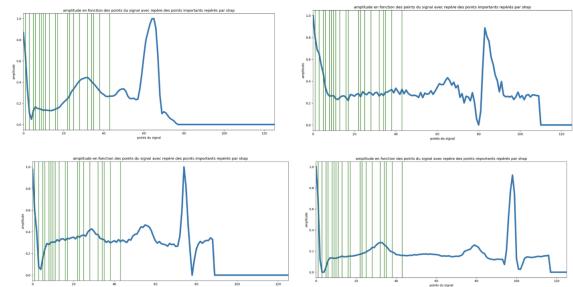


Figure 6 - LR : Exemples de valeurs importantes placées sur des ECG

Cependant, nous pouvons relever l'absence de points importants situés en fin de l'ECG malgré des caractéristiques très variables entre les signaux à cet endroit. Cela explique peut-être les faiblesses de ce modèle.

Pour conclure, ce modèle n'obtient pas de résultats très satisfaisants, en effet, les scores sur la classe 1 restent trop faibles voire aléatoires. Ceci ne convient pas dans le cadre d'un projet sur la détection d'anomalie cardiaque. La régression logistique n'est pas adaptée à un grand nombre de caractéristiques. Cet algorithme ne peut pas résoudre le problème de non-linéarité ce qui nécessite la transformation des caractéristiques non linéaires.

3.2.4 Conclusion : avantages et limites du modèle

Ce modèle de régression logistique n'est pas très optimal pour notre objectif qui porte sur la prédiction des battements cardiaques. En effet, il présente un manque de précision important sur la classe anormale, qui est celle qui nous importe le plus dans ce contexte. Le rappel sur cette classe peut être amélioré, mais cela se fait au détriment de la précision, qui est désormais comparable à celle d'une prédiction aléatoire. Dans un projet de détection d'anomalies cardiaques, il est crucial d'obtenir un modèle qui peut détecter les anomalies avec une haute précision, car toute erreur de prédiction peut avoir des conséquences graves pour la santé du patient à prévoir. Par conséquent, ce modèle de régression logistique doit être remplacé par un modèle plus sophistiqué.

3.3 Arbre de décision

L'arbre de Décision est un modèle de classification supervisé. L'arbre de décision peut être décrit comme une représentation visuelle d'un algorithme d'apprentissage supervisé qui sélectionne automatiquement les variables discriminantes à partir d'une base de données même très volumineuse. L'algorithme permet d'extraire des déterminismes (des règles logiques de cause à effet) dans le jeu de données appelés « noeuds » ou « décisions » en suivant différents critères quantitatifs. Chaque noeud (décision de séparation en deux branches) est le résultat d'un test quantitatif. Les derniers noeuds forment les groupes terminaux appelés feuilles de l'arbre, chacune ayant un des labels de la variable cible (ici 0 ou 1 pour ECG Normal ou Anormal respectivement). Pendant la phase d'entraînement, les « décisions » du modèle sont prises à partir de tests « optimaux » basés sur des seuils calculés par rapport à l'ensemble d'entraînement. En général l'entraînement s'arrête selon un critère d'arrêt, d'homogénéité des groupes ou de profondeur maximum d'arbre (nombre maximum de noeuds pour chaque branche). L'avantage de l'arbre de Décision est qu'il permet d'obtenir des résultats facilement interprétables et procure une représentation graphique des décisions de classifications successives permettant de comprendre les règles logiques qui ont abouti aux prédictions proposées par le modèle (voir exemple figure 1). La nature hiérarchique d'un arbre de décisions permet de voir facilement quels attributs sont les plus importants, ce qui n'est pas toujours clair avec d'autres algorithmes, similaires aux réseaux neuronaux.

3.3.1 Arbre de décision sans optimisation

Nous avons commencé par entraîner un arbre simple de profondeur 2 (figure 1DT).

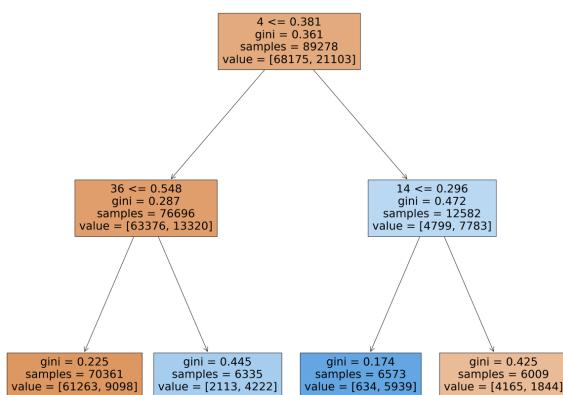


Figure 1 - DT : Arbre de profondeur 2. Le gini indique la valeur d'impureté du noeud.

La visualisation de l'arbre en fait un modèle très

compréhensible. Par exemple ici on observe que les deux premières branches sont créées à partir d'une valeur seuil de 0,381 sur la variable explicative numéro4.

Maintenant que notre modèle est entraîné nous allons faire une prédiction sur les label du jeu de données de test. La prédiction nous donne des résultats résumé dans le tableau 1-DT.

Classe	precision	rappel	f1
0	0.86	0.96	0.91
1	0.79	0.48	0.60

Tableau 1 - DT : Scores « baseline » obtenus sur un arbre de profondeur 2 sur le jeu de test.

Ces scores sont bons pour la classe 0 mais faible pour la classe 1 (minoritaire dans les données). Le rappel de la classe 1 est de moins de 50% ce qui indique que plus de la moitié des ECG Anormaux ne sont pas détectés par le modèle. Notre f1-score pour la classe 1 est de 0.60 ce qui constituera notre Baseline. Ce score est faible et nous allons tenter de l'améliorer en optimisant les hyperparamètres du modèle.

3.3.2 Arbre de décision avec optimisation

La première étape d'optimisation consistait à déterminer la profondeur optimale pour l'arbre de décision. Une grille de recherche nous indique d'utiliser des valeurs de profondeur haute. Nous testons donc le modèle avec une profondeur de 20. Les scores obtenus sont meilleurs. Cependant le modèle devient très long à produire une classification et les résultats deviennent très difficiles à interpréter, comme observé sur la figure2-DT. Un tel arbre semble peu flexible aura probablement du mal pour s'adapter à un nouveau jeu de données.

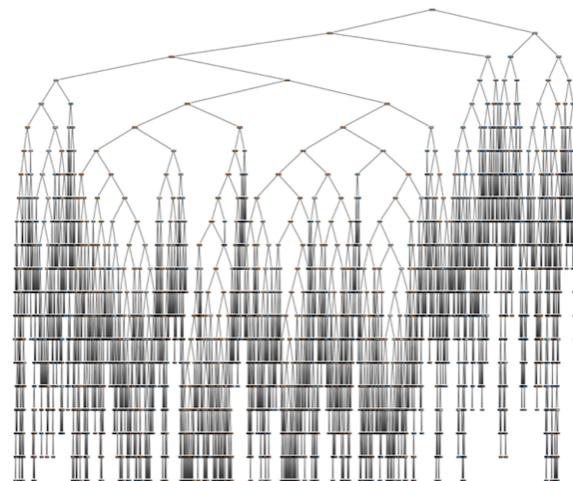


Figure 2 - DT : Arbre de profondeur 20. L'arbre très profond est devenu difficile à comprendre.

Nous avons décidé de représenter les scores d'accuracy en fonction de la profondeur de l'arbre. Et ce au cours de l'apprentissage pour visualiser la différence entre le jeu d'entraînement et le jeu de test (voir figure 3-DT)

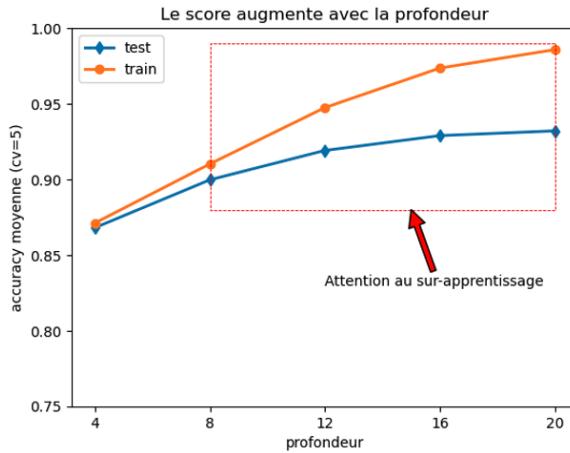


Figure 3 - DT : Accuracy en fonction de la profondeur de l'arbre.

Les résultats de la figure 3-DT indiquent que les performances du modèle augmentent avec la profondeur, cependant on observe clairement que le modèle obtient une meilleure performance sur le jeu d'entraînement par rapport au jeu de test, ce qui indique un défaut de généralisation, on parle de surapprentissage. A partir d'une profondeur de 8 nous observons du surapprentissage, nous fixeront la profondeur à 8 pour l'optimisation.

D'autres hyperparamètres peuvent être optimisés pour tenter de limiter le surapprentissage tel que 1/ le minimum d'échantillons par feuille, 2/ le minimum de diminution d'impureté par noeud, et 3/ le nombre de variables maximum utilisées par décision. En jouant avec ces paramètres, le modèle est contraint, les scores baissent très légèrement mais cela limite le phénomène de surapprentissage (données non montrées).

	Classe s	Précision		Rappel		F1-score		Rappel trop bas pour la classe 1
		0	1	0	1	0	1	
baseline profondeur = 2	0	0,86	0,96	0,91	0,96	0,91	0,91	
baseline profondeur = 2	1	0,79	0,48	0,60	0,48	0,60	0,60	Rappel trop bas pour la classe 1
		Optimisation testée	Impact sur la précision	Impact sur le rappel	Impact sur le f1-score			
1	profondeur = 8	0	0,91	0,98	0,94			
1	profondeur = 8	1	0,91	0,69	0,78			
2	profondeur = 20	0	0,95	0,95	0,95			
2	profondeur = 20	1	0,85	0,84	0,84			Surapprentissage

Tableau 2 - DT : Récapitulatif de l'optimisation

3.3.3 Analyse et Interprétation

L'arbre nous permet de connaître les variables explicatives de l'ECG avec le plus d'importance pour la

classification. Dans un premier temps nous avons pu voir que pour l'arbre simple de profondeur égale à 2, les points importants étaient 4, 14 et 36 (voir figure 1-DT). Nous nous sommes demandé si la profondeur pouvait affecter le choix des points importants par le modèle ? Pour répondre à cette question nous avons effectué une analyse de l'importance en fonction de la profondeur (figure 4-DT)

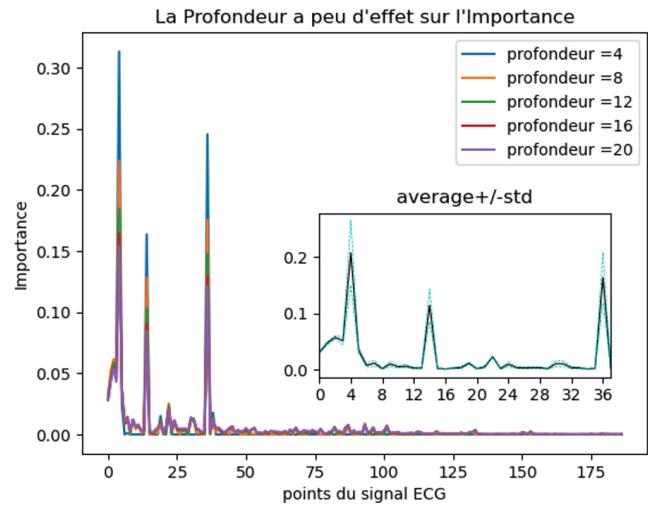


Figure 4 - DT : importance en fonction de la profondeur. On remarque que les points importants sont les mêmes quelque soit la profondeur de l'arbre. inset : moyenne +/- déviation standard de l'ensemble des profondeurs testées.

Pour finir et afin de visualiser les points importants pour la classification nous avons superposé les points importants sur le signal ECG au cours du temps (figure 5-DT).

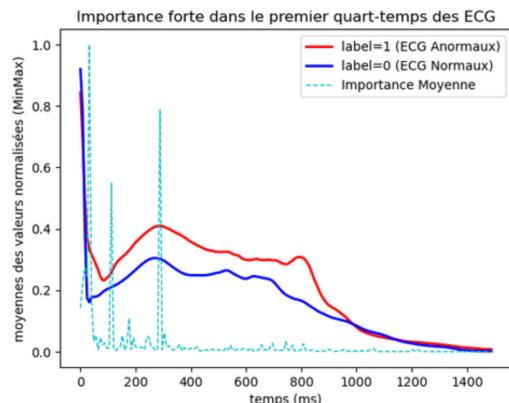


Figure 5 - DT : Importance superposée avec le signal ECG moyen. On observe que les pics d'importance correspondent aux ondes R1, S, et T. On observe que le pic R2 est noyé dans la moyenne et n'est donc pas visible.

Cette dernière analyse nous permet de donner une interprétation quant au fonctionnement de notre modèle. Dans tous les cas, le modèle préfère choisir les 5 premiers points ainsi que les points 14 et 36. La visualisation figure 5-DT nous montre que ces points correspondent successivement aux ondes R(autour de 0 ms), S (autour de 100ms) et T (autour de 300ms). La valeur prise par ces points indique si le pic attendu dans le cas normal a eu lieu ou non. Comme ces points sont situés proche du point 0 les ondes restent relativement bien alignées temporellement sur ces points, alors que les variations plus lointaines du zéro ont tendance à se désynchroniser du fait d'une variabilité inter-individuelle non négligeable. Ce phénomène est d'ailleurs visible sur nos signaux moyens qui ont leurs ondes R2 quasi invisibles.

3.3.4 Conclusion : avantages et limites du modèle

Le modèle procure une bonne explicabilité et est facile à prendre en main. Cependant le modèle semble sujet au surapprentissage dès lors que la profondeur est trop forte. Ce qui nous empêche ici d'atteindre des performances hautes surtout pour la classe 1, malgré les optimisations supplémentaires effectuées soit par rééquilibrage des classes soit par sous-échantillonnage de l'ECG (données non montrées ici). Il nous faudra donc explorer d'autres modèles ou bien complexifier l'arbre simple en utilisant un foret aléatoire (non testée ici) ou bien même un CATboost (voir plus loin).

3.4 K plus proches voisins

KNN est une méthode de classification non paramétrique qui recherche les k voisins les plus proches d'une nouvelle observation pour prédire sa classe. Contrairement à d'autres algorithmes, il ne fait pas d'hypothèses sur les données, ce qui le rend adapté aux ensembles de données complexes et hétérogènes. Pour obtenir de bons résultats avec ce modèle, il est crucial de choisir la valeur optimale de k et la distance appropriée pour mesurer la similarité entre les points de données. Une valeur de k trop petite peut entraîner un surapprentissage, tandis qu'une valeur trop élevée peut entraîner un biais significatif dans la classification. Il faut donc trouver le juste milieu pour notre modèle.

Cependant, le modèle KNN peut être coûteux en termes de temps de calcul, en particulier pour des ensembles de données volumineux ou avec de nombreuses variables. Le calcul des distances entre les nouvelles observations et l'ensemble d'apprentissage peut être gourmand en ressources. Pour résoudre ce problème, il est possible d'utiliser des techniques de réduction de dimension pour réduire la complexité des données ou d'explorer d'autres algorithmes de classification plus adaptés aux grands ensembles de données.

3.4.1 KNN sans optimisation

Voici dans un premier temps les résultats obtenus à partir du rapport de classification et de la matrice de confusion du modèle KNeighborsClassifier avec les hyperparamètres par défaut ($k=5$, $\text{weights} = \text{'uniform'}$) :

Modèle KNN avec $k = 5$					
Temps d'entraînement: 0.03 secondes					
Temps d'exécution: 59.48 secondes					
	precision	recall	f1-score	support	
0.0	0.96	0.99	0.97	18990	
1.0	0.95	0.87	0.91	5810	
accuracy			0.96	24800	
macro avg	0.96	0.93	0.94	24800	
weighted avg	0.96	0.96	0.96	24800	

Figure 1 - KNN : Tableaux des scores sans optimisation

Le modèle présente des résultats prometteurs pour la classe normale (0) avec une précision élevée. Cependant, il prédit moins bien la classe anormale (1), ce qui peut être attribué au déséquilibre des classes dans les données. En conséquence, le recall (rappel) pour la classe 0 est de 99%, tandis que pour la classe 1, il est de 87%. Cette disparité dans les performances se reflète dans le f1-score.

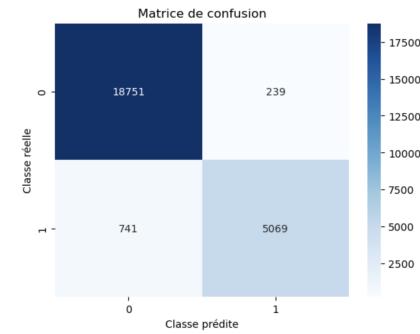


Figure 2 - KNN : Matrice de confusion du modèle sans optimisations

Afin d'améliorer le modèle, il est nécessaire d'appliquer des techniques d'optimisation et de vérifier s'il y a un surajustement (overfitting) des données. Ces mesures visent à rééquilibrer les performances du modèle entre les classes et à améliorer sa capacité à prédire avec précision la classe anormale.

3.4.2 KNN avec optimisation

Dans cette partie, nous avons utilisé la technique de réduction de dimensionnalité PCA (Principal Component Analysis) afin d'accélérer le modèle. La PCA permet de réduire la complexité des données en diminuant leur dimensionnalité, ce qui facilite le travail du modèle et rend les données plus facilement interprétables. Cette approche nous a permis de déterminer la valeur optimale de l'hyperparamètre k en utilisant une boucle avec k variant de 1 à 15. Ce processus aurait été coûteux en termes de temps et de calcul pour la machine, surtout pour des données volumineuses.

Le graphique ci-dessous représente les valeurs de k en fonction de l'exactitude (accuracy) obtenue après la réduction de dimensionnalité par PCA :

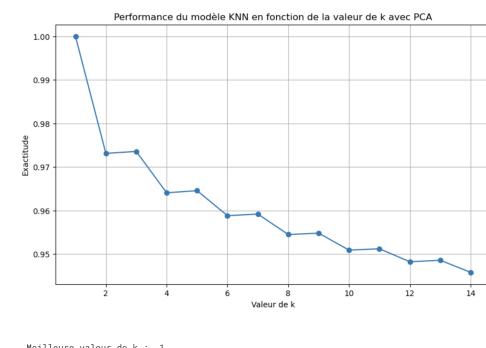


Figure 3 - KNN : Valeurs de k en fonction de l'exactitude du modèle KNN via une PCA.

Le graphique suggère que la valeur optimale de k pour maximiser l'exactitude du modèle est $k = 1$.

Cependant, il est important d'examiner de manière critique ce choix, car une valeur aussi petite de k est susceptible de conduire à un surapprentissage des données. Le surapprentissage se produit lorsque le modèle s'adapte trop étroitement aux données d'entraînement et perd sa capacité à généraliser de manière précise sur de nouvelles données.

Il est important de noter que la dimension réduite à 2, représentée dans le graphique, est utilisée à des fins de visualisation et pourrait ne pas capturer la totalité de la variance des données. Il est recommandé d'effectuer une analyse plus détaillée et de prendre en compte d'autres métriques d'évaluation pour choisir la meilleure valeur de k.

```
Modèle KNN avec k_best = 1

Temps d'entraînement: 0.02 secondes
Temps d'exécution: 81.85 secondes
precision      recall      f1-score
0.0          0.98      0.99      0.98
1.0          0.95      0.92      0.94

accuracy           0.97
```

Figure 4 - KNN : Scores avec optimisation

En utilisant k = 1 pour le modèle KNeighborsClassifier, nous avons observé une amélioration significative des résultats. Le recall pour la classe 1 a augmenté de 87% à 92%, ce qui a également entraîné une amélioration du f1-score de 91% à 94%. De plus, nous avons constaté une amélioration de la précision pour la classe 0.

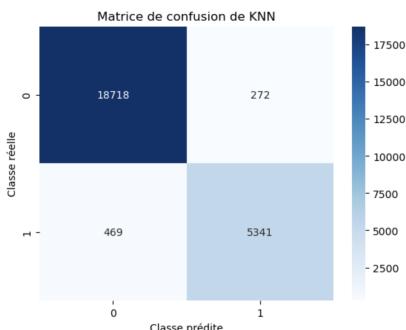


Figure 5 - KNN : Matrice de confusion du modèle avec optimisation

En examinant la matrice de confusion, nous constatons que sur les 18990 échantillons de la classe normale, 18718 sont correctement prédits (vrais négatifs), tandis que sur les 5810 échantillons de la classe anormale, 5341 sont correctement prédits (vrais positifs). Cependant, il y a également 272 échantillons de la classe anormale qui sont classés à tort comme normaux (faux négatifs) et 469 échantillons normaux classés à tort comme anormaux (faux positifs).

3.4.3 Analyse et Interprétation

La courbe d'apprentissage est un graphique qui montre l'évolution de la performance d'un modèle en fonction de la quantité de données utilisées pour l'entraînement. Elle permet de visualiser si le modèle bénéficie d'une quantité suffisante de données d'entraînement et si le modèle est en sous-apprentissage ou en sur-apprentissage.

Il est important de trouver un équilibre entre l'exactitude et la capacité du modèle à généraliser sur de nouvelles données. Dans certains cas, une valeur plus élevée de k peut être préférable pour réduire le risque de surapprentissage et permettre une meilleure généralisation.

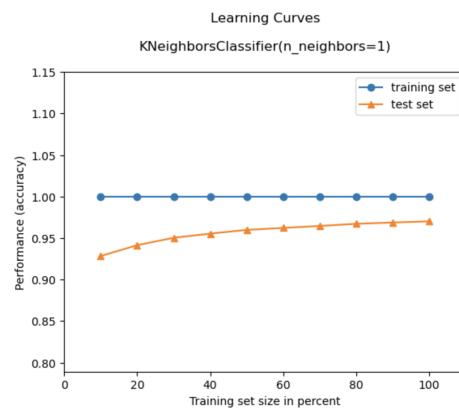


Figure 6 - KNN : Courbe d'apprentissage du modèle KNN avec k = 1.

La courbe d'apprentissage ci-dessus pour k = 1 indique un surapprentissage des données, ce qui signifie qu'il a appris les exemples d'entraînement de manière trop précise, au point de ne pas généraliser efficacement sur de nouvelles données. En conséquence, il présente une précision élevée sur l'ensemble d'entraînement, mais une performance moins satisfaisante sur l'ensemble de test. Il est donc important de prendre des mesures pour éviter l'overfitting et améliorer la capacité du modèle à généraliser sur de nouvelles observations.

Dans le cadre de ce sur-ajustement, l'algorithme d'optimisation Ridge applique la régularisation en pénalisant les valeurs extrêmes des paramètres et en favorisant les modèles plus simples qui fournissent des résultats similaires. Cela permet de contrôler la complexité du modèle et de favoriser une meilleure généralisation sur de nouvelles données. En ajustant le paramètre de régularisation, on peut trouver le bon équilibre entre précision et complexité du modèle.

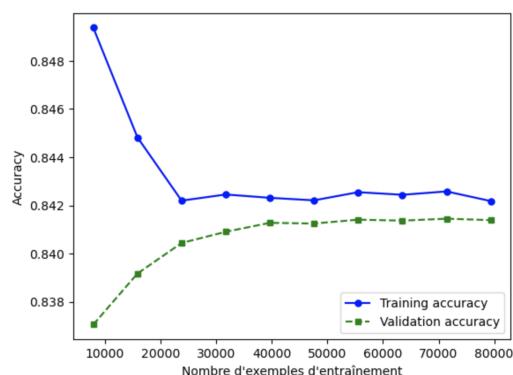


Figure 7 - KNN : Courbe d'apprentissage du modèle KNN avec régularisation Ridge et $k = 1$.

L'analyse de la courbe d'apprentissage obtenue avec la régularisation de type Ridge présente certains aspects encourageants, mais aussi des éléments qui nécessitent une évaluation plus approfondie. D'un côté, la convergence des courbes de l'ensemble d'apprentissage et de l'ensemble de validation vers une performance similaire suggère que la régularisation a contribué à réduire le surapprentissage. Cela indique que le modèle est capable de généraliser correctement sur de nouvelles données, ce qui est un résultat positif. D'un autre côté, la décroissance de la courbe de l'ensemble d'apprentissage peut indiquer que le modèle n'est pas en mesure de capturer de manière adéquate les motifs complexes des données d'apprentissage. Cela pourrait limiter ses performances et sa capacité à s'ajuster aux données réelles. Cependant, la croissance de la courbe de l'ensemble de test à mesure que la taille de l'apprentissage augmente est encourageante, suggérant que le modèle s'améliore avec plus de données.

Choisir un entier k trop petit, comme $k = 1$, dans un problème de classification peut entraîner des ambiguïtés en particulier lorsque le nombre de classes est pair. Il est généralement recommandé de choisir un nombre k impair, ce qui permet d'éviter les égalités lors du vote. De plus, le choix de k doit être suffisamment grand pour permettre une décision basée sur un nombre significatif de voisins.

En analysant le graphique de la figure 1, nous pouvons estimer un k qui pourrait être adapté à notre problème. En choisissant $k = 3$, nous avons un nombre impair, qui est suffisamment grand par rapport au nombre de classes étudiées, et qui donne une bonne exactitude du modèle. Il serait donc plus judicieux d'opter pour ce choix raisonnable dans le cas d'un problème de classification binaire.

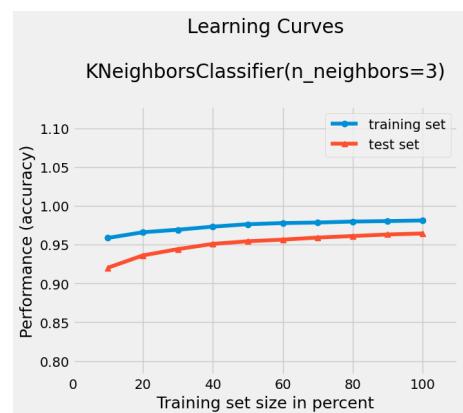


Figure 8 - KNN : Courbe d'apprentissage du modèle KNN avec $k = 3$.

Le modèle KNN avec $k = 3$ a subi une légère réduction du rappel pour cette classe minoritaire. Cependant, cette diminution est contrebalancée par une amélioration de la capacité du modèle à généraliser et donc potentiellement diminuer le surapprentissage.

Par ailleurs, les courbes d'apprentissage du jeu d'entraînement et du jeu de test présentent une augmentation simultanée et une convergence avec un écart raisonnable entre les deux. Cela indique que le modèle KNN avec $k = 3$ est capable d'apprendre efficacement à partir des données d'entraînement tout en maintenant de bonnes performances sur de nouvelles données.

3.4.4 Conclusion : avantages et limites du modèle

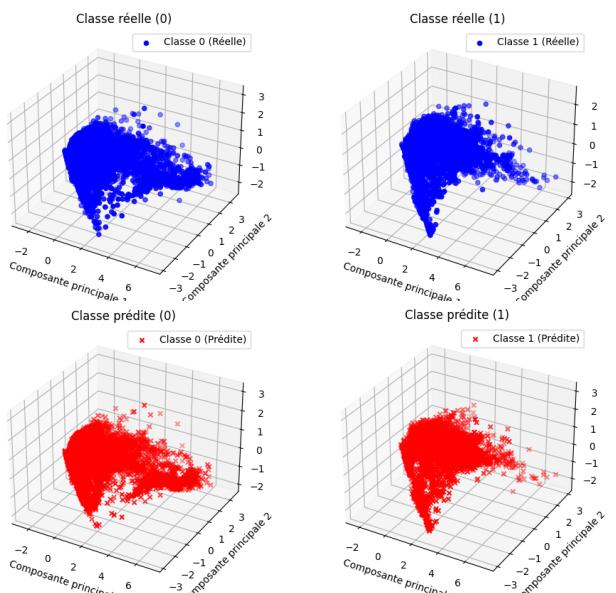


Figure 9 - KNN : Graphique 3D des classes réelles et prédictes

En analysant visuellement les graphiques 3D des classes réelles et prédictes ci-dessous, nous observons une similarité entre les zones de points. Cela indique que le modèle KNN avec $k = 3$ a réussi à capturer les caractéristiques distinctives des classes normales et anormales, ce qui suggère une certaine fiabilité dans sa capacité à distinguer ces deux types d'instances.

En conclusion, le modèle KNN est un outil simple

et efficace pour la classification des battements cardiaques. Cependant, il est important de considérer les limites liées au choix de k , de la qualité des données et aux contraintes spécifiques du modèle, notamment pour les grandes bases de données comme des ECG. Une évaluation approfondie et une comparaison avec des approches plus avancées sont recommandées pour une meilleure compréhension des performances du modèle.

4 Modèles avancés et Deep Learning

CatBoost - Artificial Neural Network (ANN) - Recurrent Neural Network (RNN)

Nous abordons maintenant l'entraînement de modèles plus complexes de type « boîtes noires » qui sont beaucoup plus difficile à comprendre. Parmi ces modèles de machine learning complexes tels que le CATboost, nous testerons aussi des modèles de deep-learning tels que les réseaux de neurones artificiels. Les modèles de deep learning présente des évolutions par rapport aux modèles de machine learning classique notamment en termes de sophistications mathématiques et conduisent à des niveaux de prédictions en général plus élevés.

4.1 Gradient Boosting : CatBoost

CatBoost est un algorithme de Machine Learning supervisé utilisé pour la classification et la régression. Il se distingue des autres méthodes basées sur les arbres de décision par sa capacité à prendre en compte les données catégorielles sans prétraitement. Il utilise une méthode appelée l'encodage ordonné pour encoder les entités catégorielles, et des arbres symétriques pour améliorer la rapidité de l'algorithme. L'algorithme CatBoost renforce la méthode d'amplification de gradient d'origine pour une implémentation plus rapide. Il conserve également certaines fonctionnalités des algorithmes précédents comme la validation croisée, la régularisation et les valeurs manquantes. De plus, il possède des paramètres de détection de surapprentissage, mais ne possède pas d'autre booster que les arbres de disponibles. CatBoost peut fonctionner avec des données de petite ou de grande taille.

4.1.1 CatBoost sans optimisation

Dans un premier temps, nous avons procédé à l'implémentation du modèle CatBoostClassifier en utilisant les hyperparamètres spécifiés dans le tableau ci-dessous. Les hyperparamètres ont été sélectionnés en ajustant leur valeur en fonction des performances du modèle afin de maximiser les métriques telles que la précision, le rappel et le f1-score.

CatBoost modèle 1	
iterations	3200
learning_rate	0.08
depth	8

Figure 1 - CAT : Sélection des hyperparamètres

Le modèle est soumis à 3200 itérations, ce qui peut paraître conséquent, mais CatBoost supporte bien les données de grande taille et garde une rapidité de calcul très avantageuse. Voici les résultats ci-dessous :

	precision	recall	f1-score	support
0.0	0.98	0.99	0.99	18990
1.0	0.98	0.93	0.95	5810
accuracy			0.98	24800
macro avg	0.98	0.96	0.97	24800
weighted avg	0.98	0.98	0.98	24800

Figure 2 - CAT : Tableau des scores sans optimisation

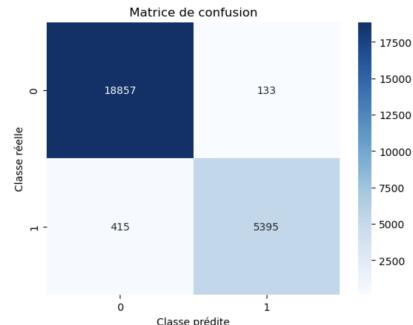


Figure 3 - CAT : Matrice de confusion du modèle sans optimisation

Le code utilise l'algorithme CatBoostClassifier pour classer des données en utilisant des hyperparamètres personnalisés tels que le nombre d'itérations, le taux d'apprentissage et la profondeur de l'arbre de décision.

Les performances du modèle sont très élevées, avec une précision de 99% pour la classe 0 et de 95% pour la classe 1. L'accuracy globale du modèle est de 98%, ce qui signifie que le modèle a correctement classé 98% des échantillons de test. Le f1-score étant suffisamment élevé pour les deux classes, signifie que le modèle réussit plutôt bien à répartir les scores entre chaque catégorie, malgré leur déséquilibre initial.

Il est à noter que l'algorithme CatBoost est connu pour avoir une capacité intrinsèque à gérer les problèmes de classes déséquilibrées, ce qui pourrait expliquer les performances élevées sans l'optimisation des hyperparamètres.

4.1.2 Catboost avec optimisation

Nous avons ajouté des hyperparamètres à notre modèle CatBoostClassifieur afin de comparer ses performances avec celles du modèle initial. En ajoutant ces hyperparamètres, nous avons cherché à affiner et à améliorer la capacité de notre modèle à capturer les caractéristiques discriminantes des données et à généraliser de manière plus précise sur de nouveaux exemples.

CatBoost modèle 2	
iterations	3200
learning_rate	0.09
depth	8
l2_leaf_reg	3
border_count	64
random_seed	42

Figure 4-CAT : Sélection des hyper-paramètres

Voici les résultats obtenus avec cette liste d'hyperparamètres :

Shrink model to first 3601 iterations.				
	precision	recall	f1-score	support
0.0	0.98	0.99	0.99	18990
1.0	0.98	0.94	0.96	5810
accuracy			0.98	24800
macro avg	0.98	0.96	0.97	24800
weighted avg	0.98	0.98	0.98	24800

Figure 5-CAT : Tableau des scores avec optimisation

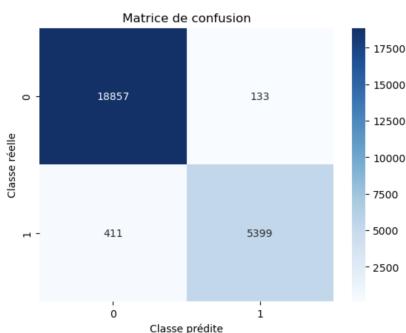


Figure 6-CAT : Matrice de confusion du modèle avec optimisation

Bien que nous ayons ajouté plusieurs hyperparamètres au modèle CatBoost, les changements observés en termes de performance ne sont pas significatifs, à l'exception d'une amélioration du rappel et du score F1 pour la classe 1.0, ce qui est potentiellement important. Il est important de noter que l'influence des hyperparamètres sur le modèle aurait pu être étudiée

au préalable afin de les ajuster de manière plus ciblée, par exemple en utilisant des techniques telles que la recherche par grille (GridSearch) ou l'algorithme Optuna, qui permettent de trouver plus rapidement les meilleurs paramètres pour le modèle. Cela aurait pu conduire à une amélioration plus significative des performances globales.

Cependant, il convient de souligner que l'ajustement des hyperparamètres ne garantit pas toujours une amélioration substantielle de la performance du modèle. Il est donc essentiel de déterminer si les modifications apportées permettent réellement d'améliorer les performances du modèle. Une évaluation plus approfondie de ces changements, notamment en comparant les résultats avec d'autres modèles ou en utilisant des métriques supplémentaires, pourrait être nécessaire pour évaluer pleinement l'impact des hyperparamètres sur la performance globale du modèle.

4.1.3 Analyse et Interprétation

Pour satisfaire la performance du modèle, il faut prendre en compte sa courbe d'apprentissage pour pouvoir estimer si notre modèle ne se trouve pas en sur-apprentissage ou sous-apprentissage, on a donc obtenu le graphique suivant :

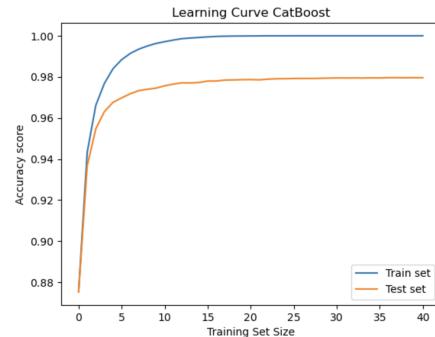


Figure 7-CAT : Courbe d'apprentissage

La courbe d'apprentissage de l'ensemble d'entraînement et de l'ensemble de test évoluent de manière similaire. Cela suggère que le modèle ne semble pas être sur-ajusté (overfitting) car il généralise bien sur les données de test. Si la courbe d'apprentissage montre que la courbe d'entraînement (training set) atteint une performance proche de 1 et que la courbe de validation (test set) atteint une performance proche de 0.98, cela peut indiquer que le modèle est capable de prédire les données de test avec une très bonne précision.

On peut s'appuyer d'autres graphiques pour analyser la robustesse du modèle, comme la courbe de calibration qui est un outil très utile pour évaluer la fiabilité des prédictions d'un modèle. Elle permet de

comparer les probabilités de prédiction avec leur taux de réussite réel. Si le modèle est bien calibré, cela signifie que les prédictions sont cohérentes avec les probabilités prédites. En revanche, si la courbe de calibration s'éloigne de la diagonale parfaite, cela indique que les probabilités de prédiction ne sont pas fiables et qu'il faut revoir le modèle ou l'entraînement des données.

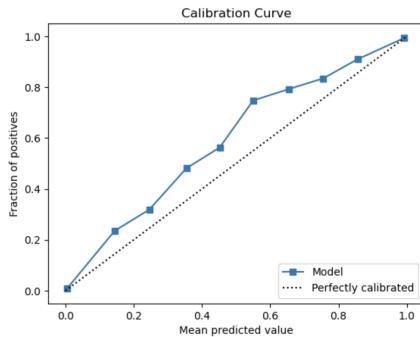


Figure 8-CAT : Courbe de calibration

On s'aperçoit que le modèle tant à se rapprocher de la courbe parfaitement calibrée, puis s'éloigne plus ou moins entre 0.5 et 0.7 (mean predicted value). Sinon en général, le modèle ne s'éloigne pas trop de la droite linéaire, ce qui permet de rajouter un second argument quant à la validité du modèle CatBoost, ainsi que de sa performance.

Le graphique de l'importance des caractéristiques (feature importance) permet de voir quelle est la contribution relative de chaque variable dans la capacité du modèle à prédire les résultats. Dans le cas de données ECG numériques, cela signifie qu'on peut observer l'importance relative de chaque mesure dans la capacité du modèle à prédire une certaine caractéristique, telle que la présence d'une anomalie ou la détection d'une pathologie particulière.

En général, une feature importance élevée indique que la variable correspondante est importante pour la prédiction et devrait être prise en compte dans l'analyse des données. À l'inverse, une faible feature importance indique que la variable n'est pas très utile pour la prédiction et peut donc être ignorée ou retirée du modèle pour simplifier sa structure.

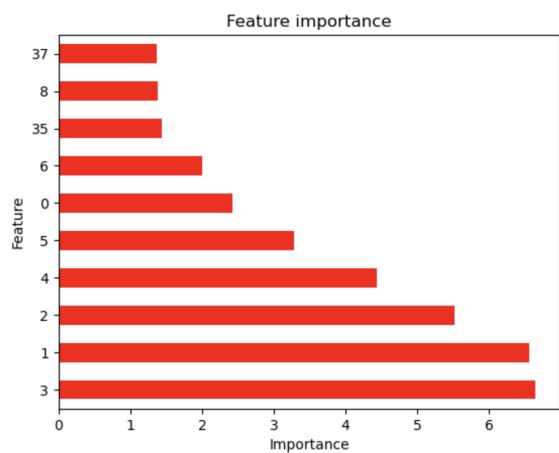


Figure 9-CAT : Importance des caractéristiques

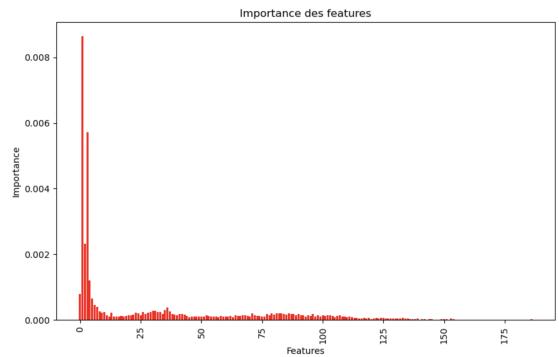


Figure 9-bis-CAT : Importance des caractéristiques

On peut constater que les variables catégorielles importantes se trouvent toutes dans les premières colonnes du dataset étudié, cela peut indiquer que ces données ont une influence plus importante sur la prédiction de la classe que les autres colonnes. À partir de la variable 37, l'importance des features commence peu à peu à diminuer puis au delà de la variable 100 cela devient pratiquement inexistant dans l'influence du modèle. Cela peut être dû à plusieurs facteurs tels que la qualité des données, la pertinence des caractéristiques pour la tâche de prédiction, etc...

Il peut être utile d'examiner de plus près les caractéristiques de ces colonnes pour comprendre pourquoi elles sont si importantes et comment cela peut aider à améliorer la performance du modèle. Par ailleurs, cela peut aussi indiquer que certaines des colonnes inutiles ou moins importantes peuvent être retirées pour réduire la dimensionnalité de l'espace des caractéristiques et faciliter le processus de modélisation tout en améliorant les performances du modèle.

4.1.4 Conclusion : avantages et limites du modèle

Le modèle CatBoost présente des avantages significatifs pour la classification binaire des battements cardiaques à partir de données ECG. Il se distingue par sa capacité à gérer automatiquement les variables catégorielles et à traiter efficacement les valeurs manquantes. De plus, il est capable de capturer les relations non linéaires entre les caractéristiques des battements cardiaques et leur classe. Le modèle CatBoost offre également de bonnes performances en termes de vitesse de calcul, ce qui en fait une option attrayante pour l'analyse des données ECG en temps réel. Toutefois, il est important de noter que l'optimisation des paramètres peut être nécessaire pour obtenir les meilleurs résultats. En résumé, le modèle CatBoost représente une approche prometteuse pour la classification des battements cardiaques, offrant à la fois précision et efficacité dans le traitement des données ECG.

4.2 Réseau de neurones Artificiel (ANN, Artificial Neural Network)

Un réseau neuronal artificiel est inspiré d'un réseau neuronal biologique lui-même composé d'un empilement de couches de neurones successives et connectées. Ainsi on peut former plusieurs couches d'algorithmes avec une architecture complexe qui vont permettre de résoudre des problèmes complexes (dans notre cas un problème de classification supervisée complexe). Nos données entrent par la couche d'entrée de l'ANN, puis sont traitées par la ou les couches intermédiaires (en fonction de l'architecture choisie) et la couche de sortie renvoie le résultat du problème. Ici on a deux neurones de sortie, un pour le label 0 (ECG Normal) et l'autre pour le label 1 (ECG Anormal).

4.2.1 ANN sans optimisation

Nous avons commencé par entraîner un réseau de neurones simple avec une seule couche intermédiaire de 20 neurones (activation relu) et avec une couche de sortie à deux neurones (activation softmax). Nous avons entraîné notre modèle sur notre jeu d'entraînement, avec 100 échantillons par batch et avec un nombre d'epoch maximum de 1000. Le modèle est muni d'une fonction de callback de type early-stopping (arrêt anticipé avec une patience de 20) qui lui permet de détecter les plateaux de performance et de s'y arrêter. Nous avons utilisé la fonction de perte 'BinaryCrossentropy', l'optimizer= 'adam' et avons suivi la métrique accuracy.

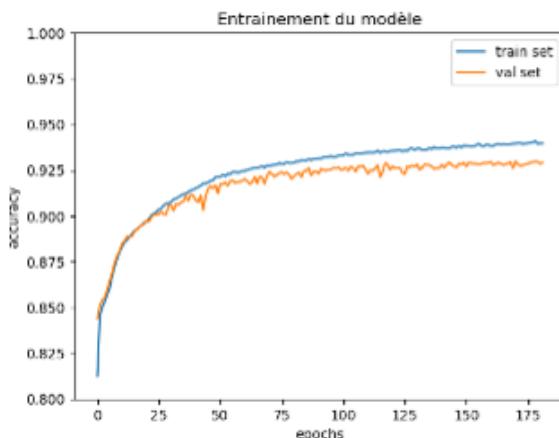


Figure 1-ANN : La courbe d'apprentissage du jeu d'entraînement atteint un plateau ce qui arrête l'entraînement au bout de 175 epochs. On remarque un sur-apprentissage qui commence tôt (epoch 25) et qui ne cesse de s'accroître.

Le suivi des scores d'accuracy lors de l'apprentissage, et la comparaison de ce suivi pour les jeux d'entraînement et de validation permet d'observer un

phénomène de sur-apprentissage (figure 1-ANN) Finalement, les scores obtenus sur notre jeu de test sont résumés sur le tableau suivant :

Classe	Precision	Rappel	f1
0	0.93	0.98	0.95
1	0.92	0.75	0.82

Tableau 1-ANN : Scores « baseline » obtenues sur le jeu de test avec un ANN à 20 neurones.

Ces scores sont déjà bien au-dessus de ceux obtenus via les modèles de machines learning classique. Cependant le score F1 de la classe 1 reste à 0.82 ce qui nécessiterait d'être améliorer pour que notre modèle commette le minimum d'erreurs possible. Nous allons donc tenter d'optimiser notre modèle afin de rehausser les scores et particulièrement ceux de la classe 1.

4.2.2 Optimisation de l'ANN via l'architecture

Le premier test a été d'identifier le nombre de neurones optimal pour l'ANN à une seule couche intermédiaire. Nous avons donc procédé à tâtons en testant différents nombres de neurones, toutes choses égales par ailleurs. Nous obtenons la courbe suivante :

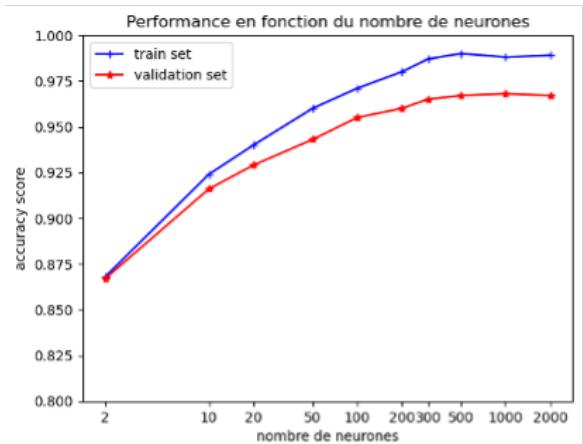


Figure 2-ANN : La performance augmente avec le nombre de neurones. Cependant, le sur-apprentissage semble aussi augmenter avec le nombre de neurones.

En augmentant le nombre de neurones nous atteignons de meilleures performances, mais nous augmentons le surapprentissage. Afin d'augmenter significativement notre score nous choisirons un nombre de neurones de 300 mais pas plus pour limiter le surapprentissage.

Nous allons maintenant observer l'effet du nombre de couches sur les performances de notre modèle à 300 neurones. Pour ce faire nous avons fait varier le

nombre de couches en conservant toutes choses égales par ailleurs. La figure 3-ANN indique que l'ajout de couches n'entraîne pas de changement notable ni d'accuracy ni de sur-apprentissage.

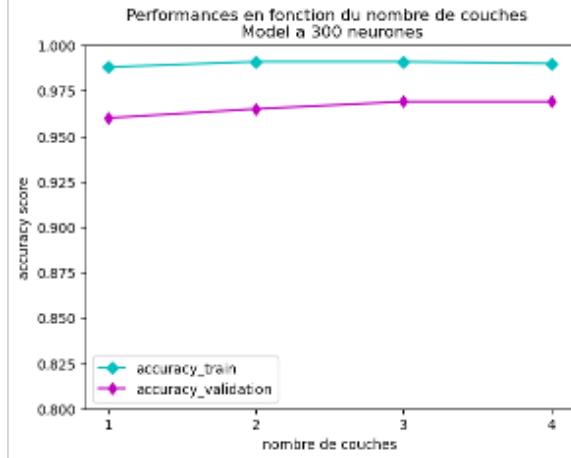


Figure 3-ANN : Les performances de notre modèle ne varient pas beaucoup en fonction du nombre de couches. Lors de l'évaluation des modèles via le jeu de test le meilleurs score F1 est obtenu par les modèles à 3 et 4 couches avec $F1 = 0.93$ pour la classe 1.

En revanche, le nombre d'epochs requises par le modèle pour s'entraîner efficacement est réduit comme le montre la figure 4-ANN.

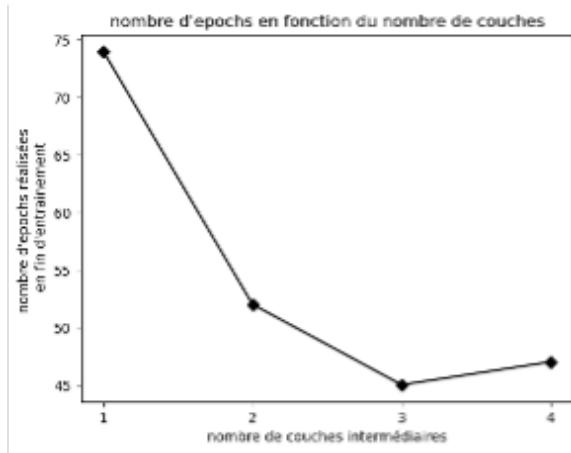


Figure 4-ANN : Ici 300 neurones sont répartis équitablement entre les couches intermédiaires du modèle. Notre modèle semble converger plus rapidement vers une solution lorsqu'il contient un plus grand nombre de couches. 3 couches intermédiaires semblent optimales dans notre cas.

Il semble que l'augmentation du nombre de couches permettrait au modèle de converger plus rapidement vers une solution. Nous optons donc pour un modèle de 300 neurones avec 3 couches intermédiaires (100 neurones par couches). L'architecture et l'entraînement

de ce nouveau modèle sont décrits figure 5-ANN. On note une convergence rapide vers des scores hauts, mais toujours avec une présence de sur-apprentissage.

Layer (type)	Output Shape
inputs (InputLayer)	[(None, 187)]
dense_layer1_relu (Dense)	(None, 100)
dense_layer2_relu (Dense)	(None, 100)
dense_layer3_relu (Dense)	(None, 100)
dense_output_softmax (Dense)	(None, 2)

Nouvelle architecture du réseau

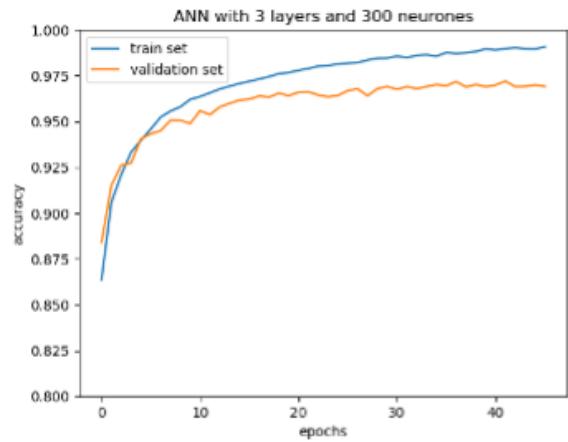


Figure 5-ANN : Notre modèles après optimisation. Comparons à la figure 1-ANN, les performances sont globalement améliorées mais le niveau de surapprentissage semble persister.

Classe	Precision	Rappel	F1-score
0	0.97 (+0.04)	0.99 (+0.01)	0.98 (+0.03)
1	0.95 (+0.03)	0.91 (+0.16)	0.93 (+0.11)

Tableau 2-ANN : Scores optimisés, obtenus sur le jeu de test avec un ANN à 3 couches intermédiaires de 100 neurones chacune. Les parenthèses indiquent la différence avec le « modèle baseline » de la figure 1-ANN. Les améliorations notables sont indiquées en rouge.

Le tableau2 nous indique de très fortes améliorations des performances après notre première étape d'optimisation. En effet nous avons gagné 16% sur notre score de rappel pour la classe1. Au total 11% gagnés sur le F1-score de la classe1 et 3% sur la classe0. L'optimisation en modifiant les caractéristiques structurales propre au réseau (nombre de neurones et nombre de couches) a porté ses fruits et nous a permis de mieux classifier les ECG de la classe Anormal.

Notons que nous avons essayé de modifier l'architecture du réseau en utilisant 150 neurones en couche1,

100 neurones en couche 2 et 50 neurones en couche 3. Cette modification n'a pas amélioré les performances, et n'a donc pas été retenue.

4.2.3 Optimisation de l'ANN via les entrées

Après avoir optimisé le modèle en modifiant ses propres caractéristiques structurales, nous décidons d'optimiser en modifiant les entrées du modèle (optimisation par les données). Pour cette étape, nous sommes passés à une taille de batch de 32 et à une patience de 10. Nous avons essayé diverses modifications notamment concernant la répartition des labels qui dans notre cas était déséquilibrée au départ. Les meilleurs résultats furent obtenus avec la fonction « RandomOverSampler » qui permet de rééquilibrer les classes en sur-échantillonnant aléatoirement des données de la classe minoritaire.

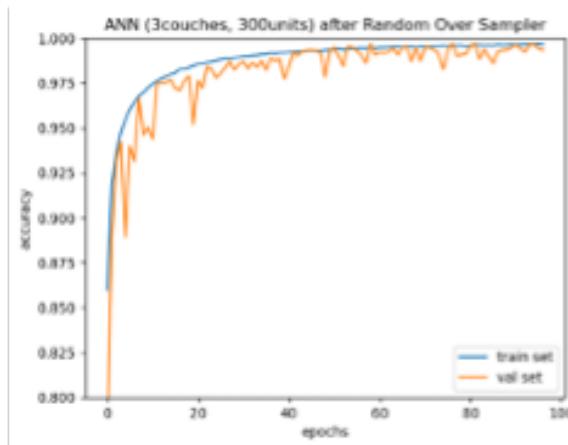


Figure 6-ANN : courbes d'apprentissage obtenues après re-échantillonnages. On observe des performances hautes et pas de sur-apprentissage.

Classe	Précision	Rappel	F1-score
0	0.98 (+0.01)	0.99 (+0.01)	0.98 (+0.00)
1	0.96 (+0.01)	0.93 (+0.02)	0.95 (+0.02)

Tableau 3-ANN : Scores optimisés obtenus sur le jeu de test après rééquilibrage des données par sur-échantillonnage aléatoire.

Le tableau 3 nous indique de petites améliorations des performances après notre deuxième étape d'optimisation. L'optimisation en modifiant les données en entrée du modèle a permis de gagner 2 points de rappel ce qui n'est pas négligeable au vu de notre problématique métier. De plus nos réglages de taille de batch et de patience, cumulés avec le rééquilibrage ont permis d'éliminer le phénomène de sur-apprentissage observé précédemment. Finalement, nous avons tenté d'injecter une variable explicative supplémentaire, la fréquence cardiaque (obtenue précédemment par feature engineering) dans notre jeu de données. Pour ce faire nous avons normalisé la fréquence cardiaque entre

0 et 1 (MinMax) ; les valeurs aberrantes supérieures à 200bpm ont été préalablement fixées à 200bpm, ainsi le 1 représente toute les valeurs supérieur ou égales à 200pbm. La colonne « fcard » a été rajoutée à $X_{train} \cup X_{test}$.

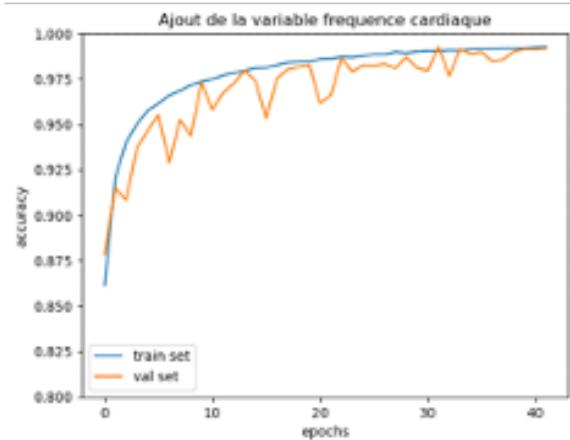


Figure 7-ANN : Courbes d'apprentissage obtenues après ajout de fcard. On observe une convergence plus rapide mais des résultats globalement similaires à ceux obtenus précédemment.

L'entraînement du modèle avec fcard en plus des optimisations précédentes n'a pas permis d'améliorer les performances.

Classe	Précision	Rappel	F1-score
0	0.98 (+0.00)	0.99 (+0.00)	0.98 (+0.00)
1	0.96 (-0.01)	0.93 (+0.00)	0.94 (-0.01)

Tableau 4-ANN : Scores obtenus sur le jeu de test après ajout de la fréquence cardiaque.

Comme le montre les résultats du tableau 4, l'ajout de notre variable fréquence cardiaque n'a pas permis d'améliorer les performances du modèle.

4.2.4 Interprétabilité et Analyse de l'ANN

A présent, nous voulons mieux comprendre comment notre modèle a fonctionné et quelles sont les variables explicatives les plus importantes pour la classification des ECG.

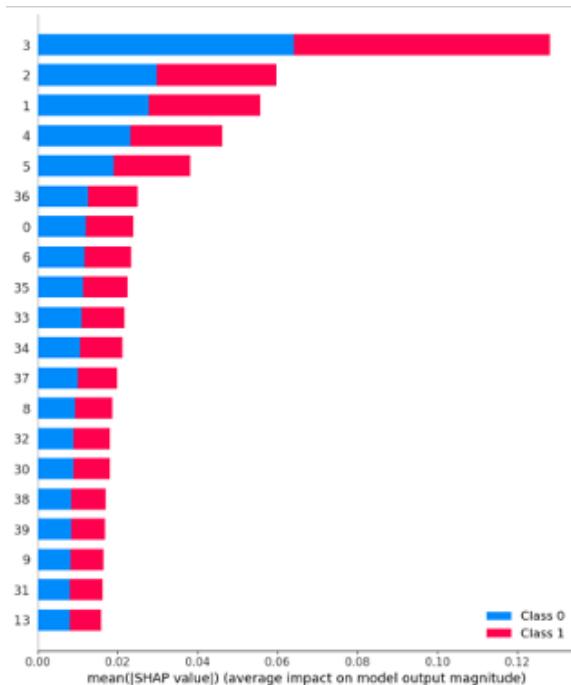


Figure8- ANN : Interprétabilité avec SHAP. Nous avons utilisé DeepExplainer de la librairie SHAP de python pour obtenir de l'explicabilité global sur notre réseau de neurones. Le modèle analysé ici est celui de la figure 7-ANN. Les valeurs shap ont été calculées sur 6% de nos données ECG du jeu de test (pour un temps de calcul d'environ 2h). On note que la variable fréquence cardiaque issue de notre étape de feature engineering n'a pas eu d'impact notable sur les performances du modèle.

L'analyse par SHAP de la figure 8-ANN nous permet de donner une interprétation globale quant au fonctionnement de notre modèle. On remarque que les 8 premiers points des courbes ECG sont ceux qui ont le plus d'impact sur les performances du modèle. Ces points correspondent au premier pique R de l'ECG. Le second groupe de point avec le plus d'importance pour la classification de notre ANN sont les points situés entre le 30ème et le 40ème points de l'ECG. Ces points correspondent à l'onde T de l'ECG. Il est intéressant de constater que comme pour l'arbre de décision vue précédemment (voir figure 5-DT) ce sont les ondes proches de l'origine qui sont les plus pertinentes pour le modèle. Les 2 ondes R1 et T semblent suffire au modèle pour résoudre la plus grande partie du problème de classification. Rien n'indique ici que le modèle utilise l'onde R2 pour la classification, qui semble trop éloignée du zéro pour conserver une position temporelle stable entre les échantillons. Nous émettons l'hypothèse que cette instabilité temporelle ne permet pas au modèle de faire des comparaisons d'une qualité statistique satisfaisante.

D'après ces résultats on pourrait penser que seuls

les 40 premiers points pourraient globalement suffire à classifier de manière satisfaisante. Pour tester cette idée nous avons effectué une réduction drastique des dimensions de l'ECG en ne conservant que les 50 premiers points de l'ECG pour l'entraînement du modèle. Les résultats ($F1\text{-score classe}0 = 0.98$ et $F1\text{-score classe}1 = 0.93$) ne sont pas très différents des résultats obtenus sur les 180 points totaux des ECG de la base de données Kaggle (voir tableau 3-ANN). Cependant la perte de 2% sur le F1 de la classe1 indique que certains points situés après le 50ème point participe tout de même à la classification mais avec un très faible impact global.

4.2.5 Conclusion : avantage et limite du modèle

Le gros avantage de l'ANN est qu'il nous permet d'obtenir des scores très hauts et tout en limitant fortement le sur-apprentissage via des réglages fin. Cependant l'ANN nous a demandé beaucoup de temps pour l'apprentissage, l'optimisation et l'interprétation du modèle en comparaison avec une régression logistique ou un arbre de décision. L'optimisation complexe de l'ANN demande un long travail de recherche et beaucoup d'essais empiriques. L'interprétabilité demande des temps de calcul très long et l'utilisation de logicielle complexe. Finalement, l'interprétation globale des résultats rejoint nos analyses précédentes obtenues sur l'arbre de décision et la régression logistique. Ce qui nous conforte dans notre interprétation globale. En conclusion, malgré ses limites, notre réseau de neurones nous a permis de surpasser les performances des modèles de machine learning classiques. Nos performances de classifications sont très bonnes avec un taux d'erreur inférieur à 5%. Néanmoins, pour notre problématique métier nous voudrions réduire au minimum possible le taux d'erreur. Pour tenter d'améliorer encore les performances nous allons tenter d'utiliser un réseau de neurones de type récurrent (voir partie suivante sur les RNN).

4.3 Réseaux de neurones récurrents

Les réseaux de neurones récurrents, ou Recurrent Neural Networks (RNN), sont des modèles d'apprentissage automatique puissants utilisés pour analyser des séquences de données telles que du texte, de la parole ou des séries temporelles. Ces réseaux sont capables de prendre en compte les informations passées grâce à des connexions récurrentes, leur permettant ainsi de se souvenir et de prendre des décisions en temps réel.

Comparés aux architectures de réseaux neuronaux classiques, les RNN sont particulièrement adaptés au traitement de données séquentielles. Cependant, ils peuvent rencontrer des difficultés lorsqu'il s'agit de gérer des dépendances à long terme, c'est-à-dire des relations entre des événements éloignés dans la séquence.

Malgré ces challenges, les RNN restent des outils précieux dans de nombreuses applications. Leur capacité à prendre en compte les informations contextuelles et à traiter des séquences en fait des candidats de choix pour des tâches telles que la traduction automatique, la génération de texte et la reconnaissance vocale.

4.3.1 RNN sans optimisation

Pour traiter le déséquilibre des classes, nous avons effectué un sur-échantillonnage en dupliquant les observations de la classe minoritaire. Ensuite, nous avons évalué différentes architectures de modèles RNN en calculant les métriques de rappel, de précision et de F1. Nous avons également tracé les courbes d'évolution de ces métriques au fil des époques.

Le modèle de deep learning que nous avons utilisé est un RNN avec une couche GRU. Nous avons ajouté une couche de Flatten pour préparer les données avant d'ajouter une dernière couche avec un neurone et une fonction d'activation sigmoïde. À chaque epoch de l'apprentissage, nous avons enregistré la valeur de la fonction de perte, l'accuracy pour le jeu de données d'entraînement et l'accuracy pour le jeu de validation.

Voici tout d'abord les résultats des métriques de ce premier modèle RNN :

	Classes	Précision	Rappel	F1-score
RNN (GRU)	0	0.99	0.98	0.99
	1	0.94	0.96	0.95

Figure 1-RNN : Tableau des scores sans optimisation

Puis, voici l'architecture de ce réseau RNN de base que nous avons utilisé :

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 187)]	0
tf.expand_dims (TFOpLambda)	(None, 187, 1)	0
gru (GRU)	(None, 187, 256)	198912
flatten (Flatten)	(None, 47872)	0
dense (Dense)	(None, 1)	47873

Total params: 246,785
 Trainable params: 246,785
 Non-trainable params: 0

Figure 2-RNN : Architecture du modèle sans optimisation

L'évolution de l'accuracy pour les données d'entraînement et de test sont présentés dans la figure ci-après :

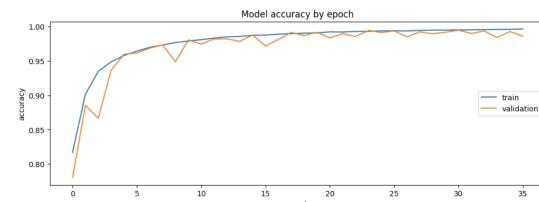


Figure 3-RNN : Evolution de l'accuracy pour les données d'entraînements et de test

Au vu de la courbe, nous constatons que la courbe des données de validation et celles d'entraînement se chevauchent et sont quasiment les mêmes, nous pouvons en déduire qu'il n'a pas de sur-apprentissage. De plus, les scores obtenus sont, à notre sens, très bons, puisqu'on a f1-score de 99% pour la classe 0 et 95% pour la classe 1.

4.3.2 RNN avec optimisation

Dans un second temps, et comme première optimisation, nous avons ajouté une nouvelle couche dense de 100 neurones dans l'espoir d'améliorer les résultats.

Voici le score des métriques obtenu avec cette première optimisation du réseau RNN :

#	Optimisation testée		Impact sur la précision	Impact sur le rappel	Impact sur le f1-score
1	GRU avec couche 100 neurones		0	0.99	0.99

Figure 4-RNN : Tableau des scores avec optimisation

Nous pouvons illustrer l'architecture du réseau RNN avec première optimisation :

Layer (type)	Output Shape	Param #	Layer (type)	Output Shape	Param #			
<hr/>								
input_5 (InputLayer)	[None, 187]	0	input_2 (InputLayer)	[None, 187]	0			
tf.expand_dims_4 (TFOpLambda)	(None, 187, 1)	0	tf.expand_dims_1 (TFOpLambda)	(None, 187, 1)	0			
a)			a)					
gru (GRU)	(None, 187, 256)	198912	lstm_1 (LSTM)	(None, 187, 256)	264192			
dense_7 (Dense)	(None, 187, 100)	25700	flatten_1 (Flatten)	(None, 47872)	0			
flatten_4 (Flatten)	(None, 18700)	0	dense_2 (Dense)	(None, 1)	47873			
dense_8 (Dense)	(None, 1)	18701	<hr/>					
<hr/>								
Total params:	243,313		Total params:	312,065				
Trainable params:	243,313		Trainable params:	312,065				
Non-trainable params:	0		Non-trainable params:	0				

Figure 5-RNN : Architecture du modèle avec optimisation

La courbe d'apprentissage obtenue est présentée sur ce graphique :

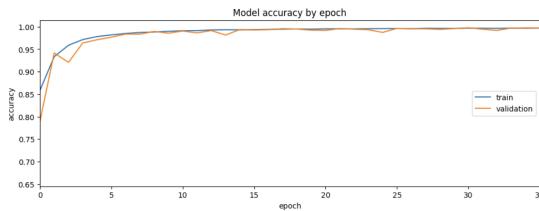


Figure 6-RNN : Courbe d'apprentissage

La courbe démontre qu'il n'y a pas de sur apprentissage où le score de la validation recoupe celui donné par les données d'apprentissage. La précision, le rappel et le score f1 sont meilleurs que le réseau de neurones sans la couche intermédiaire.

Nous avons pu effectuer une seconde optimisation où nous avons testé le très célèbre modèle LSTM très réputé pour capter l'essence du langage naturel mais également adaptable à tout type de séries temporelles.

Ce modèle obtient les scores suivants :

#	Optimisation testée	Impact sur la précision	Impact sur le rappel	Impact sur le f1-score
2	LSTM	0 1	0.98 0.97	0.99 0.94

Figure 7-RNN : Tableau des scores avec optimisation

Ce modèle à l'architecture suivante :

Figure 8-RNN : Architecture du modèle avec optimisation

L'allure de la courbe d'apprentissage est la suivante :

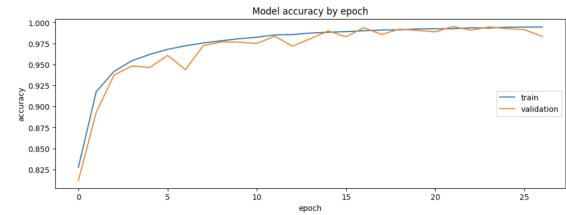


Figure 9-RNN : Courbe d'apprentissage

Les performances de ce modèle sont sensiblement les mêmes. Il est possible de remarquer que le F1 score a augmenté de 1% par rapport à la première architecture, et il n'y a toujours pas de surapprentissage.

Puis, nous avons effectué un autre modèle avec une troisième optimisation, que nous avons testé qui est le même que le précédent mais nous avons ajouté une couche dense de 100 neurones dont nous avons obtenu les scores :

#	Optimisation testée	Impact sur la précision	Impact sur le rappel	Impact sur le f1-score
3	LSTM avec couche 100 neurones	0 1	0.99 0.96	0.99 0.97

Figure 10-RNN : Tableau des scores avec optimisation

On obtient l'architecture suivante :

Layer (type)	Output Shape	Param #
<hr/>		
input_3 (InputLayer)	[None, 187]	0
tf.expand_dims_2 (TFOpLambda)	(None, 187, 1)	0
a)		
lstm (LSTM)	(None, 187, 256)	264192
dense_4 (Dense)	(None, 187, 100)	25700
flatten_2 (Flatten)	(None, 18700)	0
dense_5 (Dense)	(None, 1)	18701
<hr/>		
Total params:	308,593	
Trainable params:	308,593	
Non-trainable params:	0	

Figure 11-RNN : Architecture du modèle avec optimisation

La courbe d'apprentissage est donnée par la figure suivante :

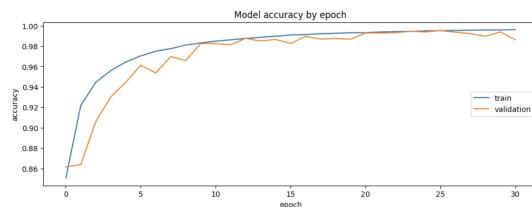


Figure 12-RNN : Courbe d'apprentissage

Le modèle donne des scores légèrement supérieurs

au modèle avec LSTM sans une couche dense intermédiaire. Cependant, il reste un peu moins performant que le modèle avec GRU et cette même couche intermédiaire.

4.3.3 Conclusion : avantages et limites du modèle

Peu importe les architectures testées en RNN, les scores peuvent s'améliorer légèrement, mais pas de façon considérable, ou se détériorer par rapport à ce qui a été présenté précédemment. Les résultats des trois modèles sont résumés dans le tableau suivant pour mettre en évidence ces constatations.

Il est important de rappeler que pour tous les modèles testés, nous avons appliqué un early stop avec une patience de 5. Cela signifie que le modèle est arrêté si l'accuracy sur l'ensemble de validation ne s'améliore pas pendant 5 époques consécutives.

De plus, il convient de souligner que les RNN demandent beaucoup de temps de calcul, ce qui nous a obligés à utiliser des cartes graphiques pour accélérer les temps d'exécution.

Comme pour les autres modèles, les performances sur la classe 1 sont moins bonnes que pour la classe 0. Cependant, les RNN obtiennent l'un des meilleurs scores parmi les modèles déjà utilisés, et légèrement plus précis que le modèle CatBoost.

5 Conclusion générale et Discussion

5.0.1 Comparaison de l'ensemble des modèles

Nous avons exploré différents modèles par niveau de complexité allant de la régression logistique jusqu'au réseau de neurones récurrent. Les tableaux suivant résument nos résultats.

Modèles Baselines	Classes	Précision	Rappel	F1-score
Régression logistique	0	0,85	0,97	0,9
	1	0,8	0,45	0,57
Arbre de décision	0	0,86	0,96	0,91
	1	0,79	0,48	0,6
KNN	0	0,96	0,99	0,97
	1	0,95	0,87	0,91
CATBOOST	0	0,98	0,99	0,99
	1	0,98	0,93	0,95
ANN	0	0,93	0,98	0,95
	1	0,92	0,75	0,82
RNN	0	0,99	0,99	0,99
	1	0,94	0,96	0,95

Figure 1-CC : Tableau des scores des modèles sans optimisation

Meilleurs modèles	Classes	Précision	Rappel	F1-score
Régression logistique	0	0,85	0,97	0,9
	1	0,8	0,45	0,57
Arbre de décision	0	0,86	0,97	0,91
	1	0,91	0,69	0,78
KNN	0	0,98	0,99	0,98
	1	0,95	0,92	0,94
CATBOOST	0	0,98	0,99	0,99
	1	0,98	0,94	0,96
ANN	0	0,97	0,98	0,98
	1	0,95	0,93	0,95
RNN	0	0,99	0,99	0,99
	1	0,97	0,97	0,97

Figure 2-CC : Tableau des scores des modèles après optimisation (ici on montre les meilleurs scores obtenus)

On remarque que dans tout les cas les modèles obtiennent de meilleurs scores pour la classe majoritaire (la classe 0 qui représente les ECG normaux) Nous avons vu précédemment que même le rééquilibrage par oversampling ne permettait pas complètement de pallier à ce biais. Ceci indique que la qualité et la quantité des données ont beaucoup d'impact sur la performance des modèles. Néanmoins, l'optimisation a permis dans certains cas de largement augmenter la performance des modèles. C'est le cas de l'arbre de décision et du réseau de neurone artificiel, notamment pour la classe 1 (minoritaire) où les performances étaient difficiles à améliorer. Le fait que l'optimisation de certains modèles n'ait pas permis d'augmenter les scores de classifications pourrait être dus à plusieurs

facteurs. L'un d'entre eux pourrait être que les paramètres du modèle était bien ajusté par défaut, ce qui entraîne un score baseline trop haute et difficile à améliorer.

De manière générale, on remarque que les modèles complexes tel que CATboost et les modèles de réseau de neurones permettent d'atteindre des scores de classifications élevé pour les deux classes. Ces modèles nous ont cependant pris beaucoup de temps à entraîner, optimiser, et interpréter. Dans certains cas les modèles à haut score sont sujet au sur-apprentissage (KNN et CATBoost). Ce sur-apprentissage indique que le modèle aura du mal à généraliser. Nous favoriserons donc les modèles pour lesquels le sur-apprentissage est faible ou inexistant. Parfois l'optimisation a permis d'éliminer ou de fortement réduire le sur-apprentissage (cas de l'ANN). Dans le cas de l'arbre si on ne réglait pas les hyperparamètres (en gardant les réglage par défaut) on obtenait un arbre très profonds avec un score très haut. Ce qui semblait très encourageant, mais en réalité le modèle était surentraîné sur nos données et était incapable de généralisé. Ce qui le rendrait inutile ou en tout cas pas éligible pour notre problème métier.

On note que pour le réseau de neurone récurrent nous n'avons pas pu trouver une façon de faire de l'interprétabilité, ni avec SHAP, ni avec SHAPASH. Bien que nous sommes conscients que cela devrait être faisable avec plus de temps, le fait de n'avoir pas pu obtenir de résultats d'interprétation de fonctionnement du modèle reflète combien il est difficile de comprendre un réseau de neurone récurrent par rapport à un modèle simple tel que la régression logistique ou l'arbre de décision.

5.0.2 Discussion et ouverture

Bien que ces modèles simple n'atteignent pas toujours des scores hauts, ils ont l'avantage de permettre de comprendre comment la classification est effectuée. Nous avons tout de même pu obtenir de l'interprétabilité pour l'ANN. Il est intéressant de constater que les parties de l'ECG utilisées par l'ANN sont les même que celles utilisées par les modèles plus simples. En effet, que ce soit pour la régression logistique, l'arbre de décision, le CATboost et l'ANN, on retrouve toujours que les points de l'ECG correspondant aux ondes R1, S et T sont les plus important pour la classification. Ceci indique encore une fois l'importance de la donnée

initiale. Le fait que les ondes P, Q et R2 qui sont les plus loin du point d'origine ne soit quasiment pas utile aux modèles indique que ces derniers ont besoin de données robustes et fiables pour faire sa prédiction. Or, dans ce cas de classification l'éloignement du point zéro est synonyme d'instabilité statistique, car les variabilités interindividuelles vont empêcher d'obtenir une trace ECG synchronisée après les 400-500ms. Cette observation est en soi très intéressante et nous donne des pistes pour une amélioration possibles de notre classification. Nous pourrions par exemple placer le zéro des courbes juste devant l'onde P et ainsi observer comment les modèles performeraient en ayant comme indicateur d'anomalie l'onde P Q R2 au lieu de R1, S, T. Nous pourrions utiliser le transfert pour re-entraîner notre modèle pre-entraîné sur les données Kaggle et l'entraîner sur les données modifiées avec un zéro avant l'onde P. Ce modèle pourrait donner des scores plus haut que ceux obtenu ici, ce qui confirmerait notre analyse.

5.0.3 Bilan global

Enfin, pour notre problématique métier nous recommanderions le modèle qui obtient le taux d'erreur le plus faible, le RNN.

6 Références

- [1] M. Kachuee, S. Fazeli and M. Sarrafzadeh. *ECG Heartbeat Classification : A Deep Transferable Representation*. IEEE International Conference on Healthcare Informatics (ICHI), pp. 443-444, doi : 10.1109/ICHI.2018.00092, 2018.