Rapport de projet

INFO4301 – Apprentissage machine

Étudiant : Nabil DAKKOUNE

Professeur : Moulay AKHLOUFI

Semestre d'automne – Année 2023/2024

Contents

Introduction 3
Exploration des données 4
Valeurs manquantes 4
Présentation des attributs 4
Statistiques et histogrammes 5
Corrélations 6
Division des données 8
Modèles 9
Points de comparaison 9
Approches 11
1. Naïve 11
2. *Avec Feature Scaling* et recherche d'hyperparamètres 13
a. Uniquement 14
b. Utilisation des caractéristiques *ejection_fraction*, *serum_creatinine* et *time* 16
c. Utilisation des caractéristiques *ejection_fraction* et *serum_creatinine* 18
d. Utilisation du sur-échantillonnage 20
e. Utilisation du sous-échantillonnage 21
f. Synthetic Minority Oversampling Technique (SMOTE) 22
3. Approche finale 23
Conclusion 25

Introduction

L'apprentissage machine est un domaine permettant de prévoir les comportements de différents phénomènes en se basant sur des cas déjà existant. L'application d'algorithmes d'apprentissage machine à la médecine en est un très bon exemple. De tels outils permettent d'épauler les médecins pour prendre des décisions vis-à-vis de diagnostics médical, de prévoir l'apparition de maladies ou encore de créer de nouveaux médicaments.

Ce projet a pour intitulé *« Utilisation de méthodes de classification pour la prédiction de survie de patients atteints d'insuffisance cardiaque »* et comme son nom l'indique, notre objectif ici est d'utiliser des modèles de classification binaire afin de prédire la survie ou la mort d'un patient atteint d'insuffisance cardiaque.

L'article de recherche intitulé *« Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone«*[1] s'est penché sur cette problématique et nous servira de support tout au long de ce projet. On peut y retrouver les approches testés, les résultats, un état de l'art, une description précise des attributs du jeu de données et bien plus.

[1] Davide Chicco, Giuseppe Jurman: "Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone". BMC Medical Informatics and Decision Making 20, 16 (2020). https://doi.org/10.1186/s12911-020-1023-5

Exploration des données

Notre jeu de données est composé de seulement 299 instances/exemples (patients) et a été récupéré sur le site archive.ics.uci.edu[2].

Valeurs manquantes

La première chose à faire lorsqu'on travaille avec un nouveau jeu de données est de savoir s'il est composé de valeurs manquantes.

```
heart_failure.info() # Affiche une description générale des données

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

Figure 1 - Informations sur les valeurs manquantes

Bingo. Notre jeu de données n'est composé d'aucune valeur manquante. Si c'était le cas, on aurait dû effectuer des choix pour continuer notre analyse (*« Doit-on remplacer les valeurs manquantes par la moyenne ? Par la médiane ? Doit-on enlever les exemples avec des valeurs manquantes ou plus violemment l'attribut en lui-même ? »*).

De plus, les attributs catégoriques (*sex*, *smoking*) sont déjà *one-hot encoded* donc tout est bon.

Présentation des attributs

D'après les informations présentes sur le site d'accès au jeu de données et à Figure 1 - Informations sur les valeurs manquantes, il est composé de 13 attributs décrits ci-dessous.

[2] https://archive.ics.uci.edu/dataset/519/heart+failure+clinical+records

- Attributs binaires :
    - *anemia* - Si le patient est atteint d'anémie - Vérifié lorsque le niveau d'hématocrite < 36%)
    - *diabetes* - Si le patient est atteint de diabète(s)
    - *high_blood_pressure* - Si le patient est atteint d'hypertension
    - *sex* – Sexe – Femme (0) ou homme (1)
    - *smoking* – Si le patient fume
    - *DEATH_EVENT* – Notre **étiquette** – Indique la survie (0) ou la mort (1) du patient dans la période qui a suivi (fenêtre d'environ 130 jours)
- Attributs numériques :
    - *age* - Âge du patient - En années - Entier entre 40 et 95
    - *creatinine_phosphokinase* - Créatine kinase dans le sang - En microgramme par litre - Entier entre 23 et 7861
    - *ejection_fraction* - Pourcentage de sang quittant le cœur à chaque contraction - En pourcentage - Entre 14 et 80
    - *platelets* - Plaquettes dans le sang - En kiloplaquette par millilitre - Flottant entre 25.01 et 850
    - *serum_creatinine* - Créatinine sérique - En milligramme par décilitre - Flottant entre 0.5 et 9.4
    - *serum_sodium* - Sodium sérique - En milliéquivalent par litre - Entier entre 114 et 148
    - *time* - Temps de suivi du patient - En jours - Entier entre 4 et 285

Statistiques et histogrammes

Voyons voir comment se comportent (d'un point de vue statistique) nos données en utilisant la méthode *describe()* de *Pandas* et *hist()* de *Matplotlib*.

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.00000	299.000000	299.000000	299.00000	299.000000	299.00000
mean	60.833893	0.431438	581.839465	0.418060	38.083612	0.351171	263358.029264	1.39388	136.625418	0.648829	0.32107	130.260870	0.32107
std	11.894809	0.496107	970.287881	0.494067	11.834841	0.478136	97804.236869	1.03451	4.412477	0.478136	0.46767	77.614208	0.46767
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.000000	0.50000	113.000000	0.000000	0.00000	4.000000	0.00000
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500.000000	0.90000	134.000000	0.000000	0.00000	73.000000	0.00000
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000.000000	1.10000	137.000000	1.000000	0.00000	115.000000	0.00000
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500.000000	1.40000	140.000000	1.000000	1.00000	203.000000	1.00000
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.000000	9.40000	148.000000	1.000000	1.00000	285.000000	1.00000

Figure 2 - Statistiques liées aux données

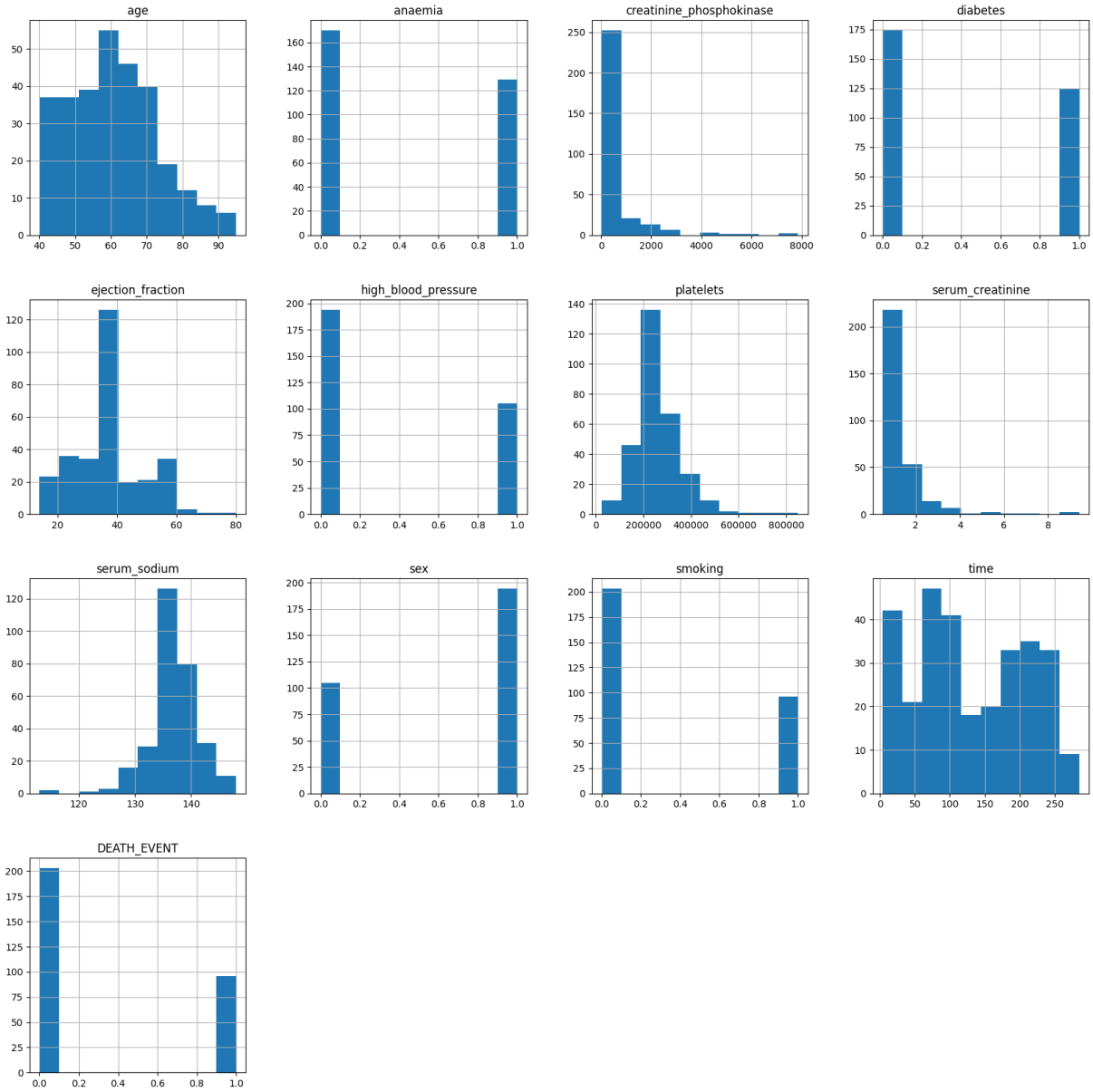

Figure 3 - Histogrammes des caractéristiques

On peut remarquer quelques informations intéressantes. L'âge moyen des patients est de 61 ans, 42% patients sont atteints de diabète(s), 65% sont des hommes ou encore que 32% fument. On remarque aussi que les valeurs possèdent des échelles très différentes et donc que qu'une **mise à l'échelle** (***feature scaling***) serait probablement favorable afin d'améliorer les performances des modèles.

Notons aussi un déséquilibre entre le nombre de patient ayant survécus (68%) et décédés (32%). On peut donc qualifié le *dataset* de **débalancé**.

Corrélations

Regardons si des attributs du jeu de données sont corrélés. Pour ce faire, on utilisera la matrice de corrélation des caractéristiques.

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
age	1.000000	0.088006	-0.081584	-0.101012	0.060098	0.093289	-0.052354	0.159187	-0.045966	0.065430	0.018668	-0.224068	0.253729
anaemia	0.088006	1.000000	-0.190741	-0.012729	0.031557	0.038182	-0.043786	0.052174	0.041882	-0.094769	-0.107290	-0.141414	0.066270
creatinine_phosphokinase	-0.081584	-0.190741	1.000000	-0.009639	-0.044080	-0.070590	0.024463	-0.016408	0.059550	0.079791	0.002421	-0.009346	0.062728
diabetes	-0.101012	-0.012729	-0.009639	1.000000	-0.004850	-0.012732	0.092193	-0.046975	-0.089551	-0.157730	-0.147173	0.033726	-0.001943
ejection_fraction	0.060098	0.031557	-0.044080	-0.004850	1.000000	0.024445	0.072177	-0.011302	0.175902	-0.148386	-0.067315	0.041729	-0.268603
high_blood_pressure	0.093289	0.038182	-0.070590	-0.012732	0.024445	1.000000	0.049963	-0.004935	0.037109	-0.104615	-0.055711	-0.196439	0.079351
platelets	-0.052354	-0.043786	0.024463	0.092193	0.072177	0.049963	1.000000	-0.041198	0.062125	-0.125120	0.028234	0.010514	-0.049139
serum_creatinine	0.159187	0.052174	-0.016408	-0.046975	-0.011302	-0.004935	-0.041198	1.000000	-0.189095	0.006970	-0.027414	-0.149315	0.294278
serum_sodium	-0.045966	0.041882	0.059550	-0.089551	0.175902	0.037109	0.062125	-0.189095	1.000000	-0.027566	0.004813	0.087640	-0.195204
sex	0.065430	-0.094769	0.079791	-0.157730	-0.148386	-0.104615	-0.125120	0.006970	-0.027566	1.000000	0.445892	-0.015608	-0.004316
smoking	0.018668	-0.107290	0.002421	-0.147173	-0.067315	-0.055711	0.028234	-0.027414	0.004813	0.445892	1.000000	-0.022839	-0.012623
time	-0.224068	-0.141414	-0.009346	0.033726	0.041729	-0.196439	0.010514	-0.149315	0.087640	-0.015608	-0.022839	1.000000	-0.526964
DEATH_EVENT	0.253729	0.066270	0.062728	-0.001943	-0.268603	0.079351	-0.049139	0.294278	-0.195204	-0.004316	-0.012623	-0.526964	1.000000

Figure 4 - Matrice de corrélation

On remarque ici quelques corrélations linéaires intéressantes entre :

- le temps de suivi et la mort (~-0.53)
- l'âge et la mort (~0.25)
- la fraction d'éjection et la mort (~-0.27)
- le sérum créatinine et l'âge (~0.16)
- le sérum créatinine et le sodium sérique (~-0.19)
- le sérum créatinine et la mort (~0.29)
- le sodium sérique et la mort (~-0.20)
- le sexe et le tabagisme (~0.45) - À prendre avec du recul car le taux homme/femme initial est disproportionné

Les graphiques étant plus simple à interpréter que de simples nombres, traçons les boîtes à moustaches de certains couples d'attributs.

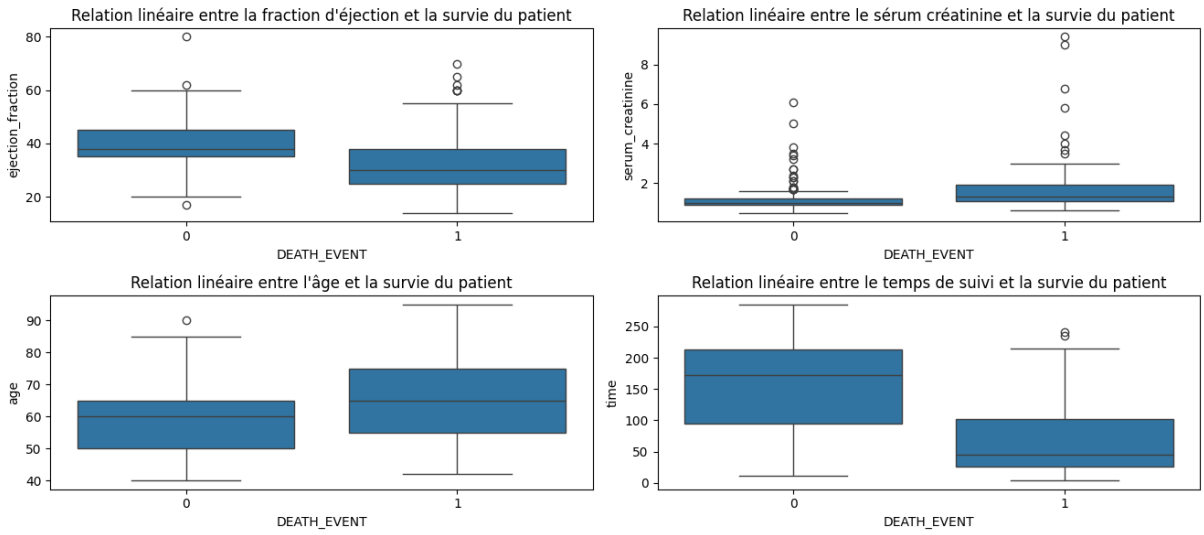

Figure 5 - Boîtes à moustaches entre attributs corrélés

On remarque bien les corrélations linéaire entre certains attributs et on peut récupérer les informations suivantes après interprétation :

- Le temps de suivi du patient impacte la survie des patients. Plus ce temps est court, plus les patients ont tendance à mourir.
- Les patients les plus âgé ont plus tendance à mourir que les patients plus jeunes.
- La fraction d'éjection (i.e. le pourcentage de sang quittant le cœur à chaque contraction) influence la survie. Plus le cœur des patients est capable de pomper de sang, moins ils ont tendance à mourir.
- La créatinine sérique agit sur la survie des patients. Un patient avec un taux élevé a plus tendance à mourir.

Premièrement ces informations vérifient les déclarations de à l'article de recherche : le triplet (*ejection_fraction, serum_creatinine* et *time*) semblent être les trois caractéristiques les plus importantes pour cette tâche de prédiction de survie de patients. Deuxièmement, on peut supposer que l'unique utilisation de ces attributs pourraient porter ses fruits et par conséquent nous tenterons une approche en les utilisant (même en sachant, et ce grâce à l'article, que c'est une très bonne idée).

Division des données

Pour nos approches, nous pourrions nous limiter à une simple séparation de l'ensemble de données en 20%/80% de façon aléatoire, mais nous allons aller plus loin. Notre ensemble de données étant très petit, l'ajout d'aléatoire dans le choix des données de test et d'entraînement résulterait en un **biais d'échantillonnage** et par conséquent, des résultats moins justes.

La stratégie proposée est celle de l'**échantillonnage stratifiée**. Cela consiste à séparer notre ensemble de données suivant un de nos attributs. Par exemple, nous savons que 65% des patients sont des hommes et donc nous pourrions faire en sorte d'obtenir un ensemble de test et d'entraînement qui respecte (plus ou moins) ce ratio.

Nous utiliserons l'étiquette ***DEATH_EVENT***. Ce choix est fait de manière réfléchie (pour que le modèle s'entraîne sur un ratio équivalent à l'ensemble de données initial). Notons qu'on aurait aussi pu stratifier par rapport à d'autres attributs (par exemple *smoking* car on aurait tendance à dire qu'un fumeur ait plus de chance de développer des maladies liées au cœur, et par conséquent, minimiser le biais d'un tel attribut pourrait se révéler pertinent).

Cette stratégie est adoptée en passant l'attribut *stratify* (de la fonction *train_test_split* de *Scikit-Learn*) avec *y* (i.e. l'objet *Pandas.core.series.Series* représentant nos étiquettes).

```python
def split_and_scale(X, y, scaling=True, stratify=None, random_state=RANDOM_SEED):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=stratify, random_state=random_state)

  if scaling:
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
  y_train = y_train.to_numpy()
  y_test = y_test.to_numpy()

  return X_train, X_test, y_train, y_test
```

Figure 6 - Fonction utilitaire qui appelle train_test_split pour la division des données

```python
X_train_default, X_test_default, y_train_default, y_test_default = split_and_scale(X, y, scaling=None, stratify=y)
```

Figure 7 - Exemple d'appel

Modèles

Les modèles utilisés sont les suivants :

- La régression logistique – *LogisticRegression* sous *Scikit-Learn* – Un modèle « classique » pour une classification binaire
- Les forêts d'arbres de décision – *RandomForest* sous *Scikit-Learn* – Un modèle qui utilise un ensemble d'arbres de décisions pour minimiser la sensibilité d'un seul arbre
- Les arbres boostés – *XGBClassifier* sous *xgboost* – Un modèle dit « *Gradient Boosted* » où un arbre est réentraîné plusieurs fois en assignant des poids aux instances mal classifiées lors des entraînements préalables
- Les machines à vecteurs de support – *SVC* sous *Scikit-Learn* – Un modèle se basant sur l'utilisation de vecteurs de supports pour créer une frontière (hyperplan) afin d'effectuer ses prédictions

Points de comparaison

La première question à se poser pour pouvoir comparer nos résultats est *« Quel niveau de performance peut-on espérer avoir ? »* et il y a plusieurs réponses répondant à cette question.

La première est se baser sur le **niveau de performance humain** (i.e. *« Quelle est la précision de l'humain sur cette tâche ? »*) et d'après l'étude liée à l'ensemble de données, la performance de spécialistes sur cette tâche est de **75%**.

Un autre niveau de performance sur lequel nous pouvons nous baser est celui des performances des **meilleurs modèles** d'apprentissage automatique sur cette tâche. Une fois de plus, l'étude nous propose de telles mesures (voir ci-dessous).

Une autre question à se poser est *« Quelle métrique est la plus cohérente vis-à-vis de notre problème ? »*. En effet, doit-on uniquement essayer d'obtenir la meilleure précision et le meilleur rappel ou autre chose ? Une telle réponse requiert l'aide de **cardiologues spécialisés** car le problème n'est plus informatique mais médical. Pour y répondre nous devons poser des questions du type *« Que se passe-t-il si un patient atteint d'insuffisance cardiaque n'est pas traité ? »*.

Dans ce cas si la réponse des spécialistes est *« Ce n'est pas très grave, la vie du patient n'est pas en danger et le traitement est cher et douloureux. »*, alors nous aurions tendance à prioriser une **haute précision** (i.e. être presque sûr que les patient détectés comme atteints le sont vraiment) du fait que **les patients non traités ne sont pas en danger**.

Au contraire, si la réponse était *« C'est potentiellement mortel pour la vie du patient s'il n'est pas traité et le traitement n'est pas si cher et si douloureux. »*, nous aurions tendance à prioriser un **haut rappel** (i.e. détecter plus de patients probablement atteint malgré que plus de patients non atteints vont aussi être détectés).

Vu que nous n'avons pas des cardiologues sous la main et que l'étude ne mentionne aucune information liées aux traitements, nous nous focaliserons sur le MCC et l'exactitude.

Table 4 Survival prediction results on all clinical features – mean of 100 executions

Method	**MCC**	F_1 score	Accuracy	TP rate	TN rate	PR AUC	ROC AUC
Random forests	**+0.384***	0.547	0.740*	0.491	0.864	0.657	0.800*
Decision tree	**+0.376**	0.554*	0.737	0.532*	0.831	0.506	0.681
Gradient boosting	**+0.367**	0.527	0.738	0.477	0.860	0.594	0.754
Linear regression	**+0.332**	0.475	0.730	0.394	0.892	0.495	0.643
One rule	**+0.319**	0.465	0.729	0.383	0.892	0.482	0.637
Artificial neural network	**+0.262**	0.483	0.680	0.428	0.815	0.750*	0.559
Naïve bayes	**+0.224**	0.364	0.696	0.279	0.898	0.437	0.589
SVM radial	**+0.159**	0.182	0.690	0.122	0.967	0.587	0.749
SVM linear	**+0.107**	0.115	0.684	0.072	0.981*	0.594	0.754
k-nearest neighbors	**-0.025**	0.148	0.624	0.121	0.866	0.323	0.493

Figure 8 - Résultats de l'étude en utilisant toutes les caractéristiques

Table 9 Survival prediction results on serum creatinine and ejection fraction – mean of 100 executions

Method	**MCC**	F_1 score	Accuracy	TP rate	TN rate	PR AUC	ROC AUC
Random forests	**+0.418***	0.754*	0.585*	0.541	0.855*	0.541	0.698
Gradient boosting	**+0.414**	0.750	0.585*	0.550*	0.845	0.673*	0.792*
SVM radial	**+0.348**	0.720	0.543	0.519	0.816	0.494	0.667

Figure 9 - Résultats de l'étude en utilisant uniquement deux caractéristiques

Table 11 Survival prediction results including the follow-up time – mean of 100 executions

Method	MCC	F_1 score	Accuracy	TP rate	TN rate	PR AUC	ROC AUC
Logistic regression (EF, SR, & FU)	**+0.616***	0.719*	0.838*	0.785*	0.860*	0.617*	0.822*
Logistic regression (all features)	**+0.607**	0.714	0.833	0.780	0.856	0.612	0.818

Figure 10 - Résultats de l'étude en utilisant uniquement trois caractéristiques

Voici ce qu'on peut remarquer :

- Les résultats varient fortement selon les caractéristiques utilisées, les meilleurs résultats étant issus de trois caractéristiques (*ejection_fraction, serum_creatinine, time*) qui, pour rappel, ont montré de fortes corrélations linéaires avec la survie des patients (Figure 5 - Boîtes à moustaches entre attributs corrélés).
- Les résultats de l'étude sont focalisés sur le MCC (i.e. le **Coefficient de Corrélation de Matthews**).
- Les résultats du modèle des k plus proches voisins montrent des résultats plus mauvais qu'un classificateur aléatoire.
- Le plus haut MCC est de 0.616 et la plus haute exactitude est de 0.833 pour le modèle de régression logistique sur trois caractéristiques.

Approches

Dans cette section nous verrons les approches testées.

1. Naïve

Dans cette partie nous effectuons ce que j'appelle un entraînement « naïf » : nous n'effectuons aucun pré-traitement sur les données et nous utilisons les paramètres par défaut des modèles de *Scikit-Learn*.

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.599	0.688	0.833	0.846	0.579	0.765
RandomForestClassifier	0.600	0.706	0.833	0.800	0.632	0.779
XGBClassifier	0.599	0.688	0.833	0.846	0.579	0.765
SVC	0.000	0.000	0.683	0.000	0.000	0.500

Figure 11 – Tableau de métriques – Approche naïve

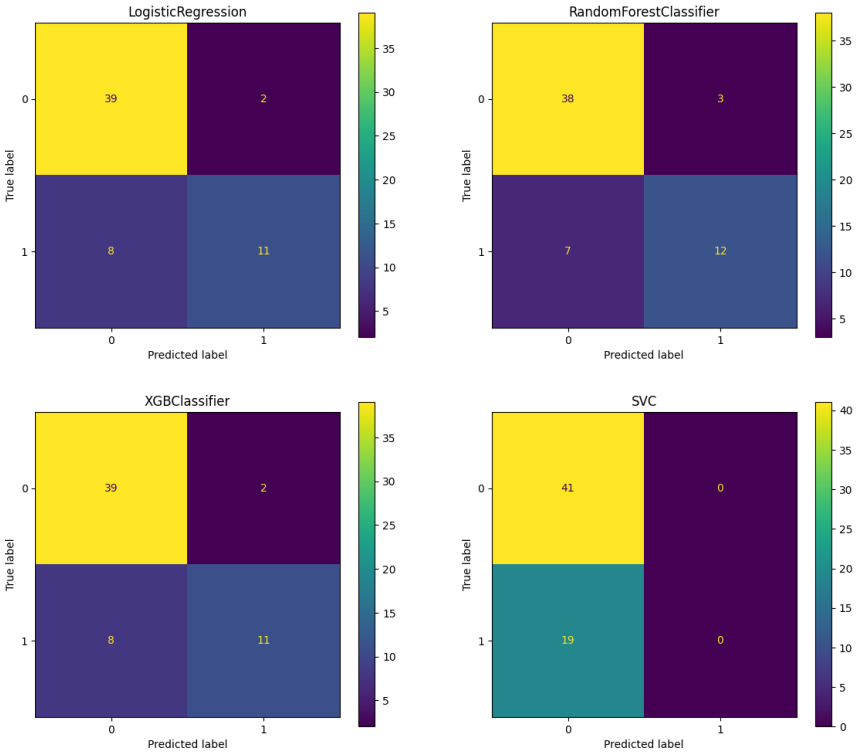

Figure 12 - Matrice de confusion - Approche naïve

La première chose qui saute aux yeux lorsqu'on regarde les métriques et les matrices de confusion est que la machine à vecteur de support prédit *y=0* en permanence. Cette conséquence est probablement due au fait que les caractéristiques n'ont pas été mis à l'échelle et qu'aucun paramètre n'a été fourni. Ce modèle est inutilisable en l'état.

Plus généralement, on remarque que tous les modèles ont plus de mal à détecter les instances positives que les instances négatives et ce, malgré un taux de positifs plus grand dans le jeu de données. Ce phénomène s'explique surement par le fait que l'ensemble de test garde le ratio 2:1 (des étiquettes) énoncé précédemment et que par conséquent, il n'est pas plus performant pour détecter l'une des deux classes.

Les autres modèles obtiennent environ les mêmes résultats (MCC :0.600, exactitude :0.833) malgré aucun prétraitement sur les données.

Certains de ces résultats peuvent s'expliquer. La première est que l'instanciation par défaut des modèles de *Scikit-Learn* est optimisée pour généraliser le plus possible et nos modèles en tirent profit. La deuxième raison quant à elle est plus étrange lorsqu'on sait que les arbres ne sont pas sensibles aux transformations et donc que la mise à l'échelle des caractéristiques ne changeraient pas (significativement) les résultats de ceux-ci.

2. *Avec Feature Scaling* et recherche d'hyperparamètres

Pour toutes les prochaines approches, on effectuera un *Feature Scaling* (centrée-réduite) et une recherche d'hyperparamètres avec *GridSearch* dont les grilles des modèles sont commentées ci-dessous.

```
log_reg_param_grid = {
    'max_iter': [25,50,100,200,400,800],  # Nombres d'itérations maximum
    'C': [0.01,0.1,1.,10,100],  # Intensités de régularisation
    # liblinear est 'un bon choix' pour les petits datasets selon la documentation sklearn
    'solver': ['lbfgs','liblinear'],  # Optimiseurs
    'class_weight': [None,'balanced']  # Modèles pénalisés (i.e. [modèle non pénalisé, modèle pénalisé])
}
```

Figure 13 - Grille de paramètres de la régression logistique

```
rand_for_param_grid = {
    'n_estimators': [25,50,100],  # Nombres d'arbres
    'criterion': ['gini','entropy'],  # Fonction de séparation des noeuds (mesure la qualité)
    'max_depth': [5,10,25],  # Profondeurs maximales
    'min_samples_split': [2,5,10],  # Nombres d'exemples minimum dans un noeud pour continuer à split
    # Modèles pénalisés (i.e. [
    #     modèle non pénalisé,
    #     modèle pénalisé selon le taux de présence initial des classes,
    #     modèle pénalisé selon le taux de présence des classes du sous-arbre actuel])
    'class_weight': [None,'balanced','balanced_subsample']
}
```

Figure 14 - Grille de paramètres de la forêt d'arbres de décision

```
xgb_param_grid = {
    'eta': [0.2,0.4,0.8,2],  # Taux d'apprentissage
    'gamma': [0,5,10,20],  # Réduction minimum du coût pour continuer de split
    'max_depth': [2,3,4,6],  # Profondeur maximale
    'lambda': [0.1,1,10,50]  #  Régularisation
}
```

Figure 15 - Grille de paramètres de l'arbre boosté

```
svc_param_grid = {
    'C': [0.01,0.1,1,10],  # Régularisation
    'kernel': ['linear','poly','rbf'],  # Noyau
    'degree': [2,3,4],  # Degrée
    'gamma': ['scale','auto'],  # Coefficient du noyau
    'class_weight': [None,'balanced']
}
```

Figure 16 - Grille de paramètres de la machine à vecteurs de support

On notera aussi que les *GridSearch* ont pour stratégie de cross-validation (i.e. l'argument *cv*) un même objet *StratifiedKFold* qui nous permet de diviser notre ensemble de données en préservant le taux de présence initial des classes.

```
skf = StratifiedKFold(n_splits=5, random_state=RANDOM_SEED, shuffle=True)
```

Figure 17 - Stratégie de séparation des ensembles d'entraînement et de validation

a. Uniquement

On essayera ici une approche « classique » : on effectue uniquement un *Feature Scaling* et une recherche d'hyperparamètres avec *GridSearch*.

On remarque quelques changement comparé aux paramètres obtenus précedemment :

```
{'C': 0.01, 'class_weight': None, 'max_iter': 25, 'solver': 'liblinear'}
{'class_weight': None, 'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 200}
{'eta': 0.2, 'gamma': 5, 'lambda': 1, 'max_depth': 2}
{'C': 1, 'class_weight': None, 'degree': 2, 'gamma': 'scale', 'kernel': 'linear'}
```

Figure 18 - Paramètres obtenus

Notons :

- *Logistic Regression* :
  - *max_iter* est au minimum
  - *C* est au minimum
- *Random Forest* :
  - *entropy* a été choisi face à *gini*
  - *n_estimators* est au maximum
  - *min_samples_split* est au maximum
- *XGBClassifier* :
  - *eta* est au minimum
  - *max_depth* est au minimum
- *SVM* :
  - *linear* est le noyau choisit
- *class_weights* mis à *None* pour tous les modèles

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.555	0.645	0.817	0.833	0.526	0.739
RandomForestClassifier	0.556	0.667	0.817	0.786	0.579	0.753
XGBClassifier	0.561	0.621	0.817	0.900	0.474	0.725
SVC	0.465	0.552	0.783	0.800	0.421	0.686

Figure 19 - Résultats obtenus - Approche Feature Scaling et GridSearch

Les métriques sont intrigantes ici. Nos trois premiers modèles possèdent la même exactitude et les résultats sont légèrement moins bons par rapport à l'approche naïve. Le peu d'exemples lors de l'entraînement implique peut-être un mauvais choix de paramètres lors des *GridSearch*. Nous regarderons comment se comportent ces paramètres lorsque d'avantages d'exemples seront présent avec l'aide de techniques de rééchantillonnage.

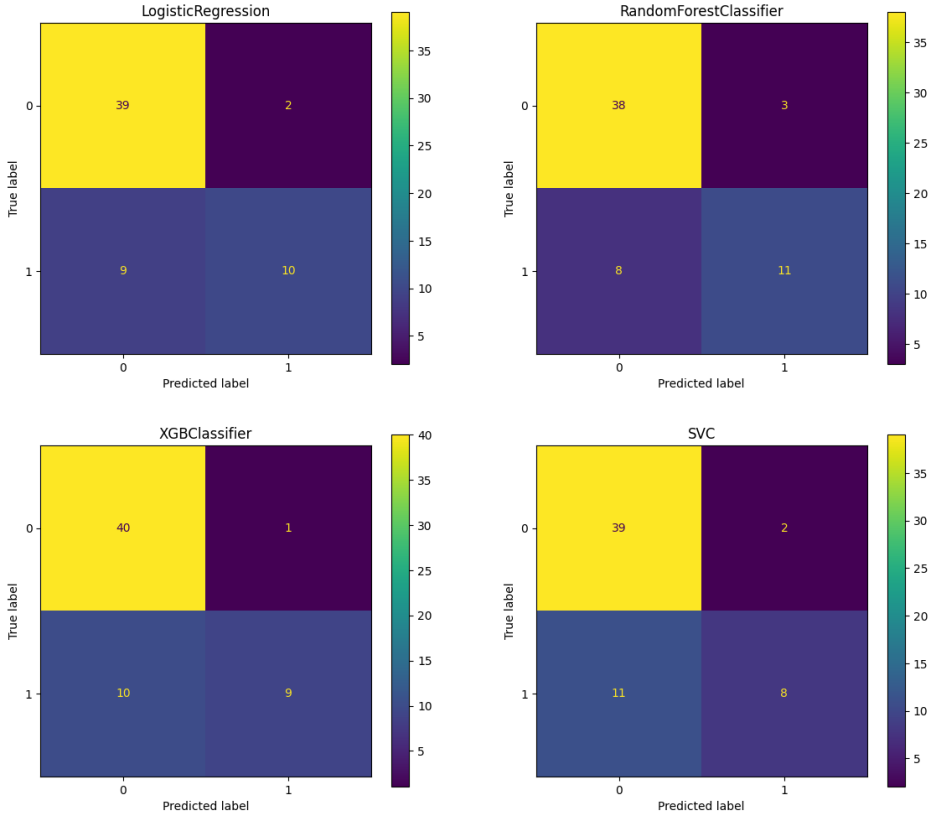

Figure 20 – Matrice de confusion - Approche Feature Scaling et GridSearch

Cette fois-ci c'est bien clair : nos modèles ont nettement plus de mal à prédire la classe majoritaire (*y = 0*).

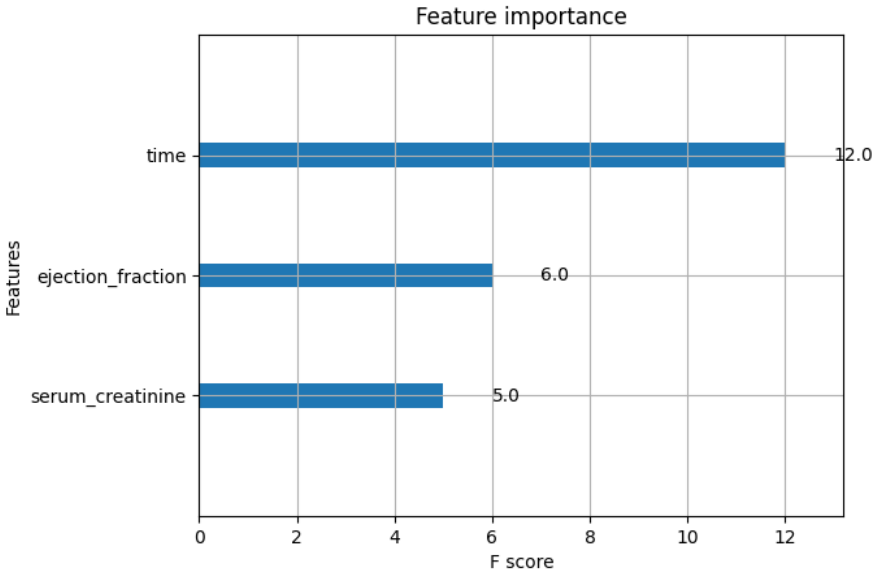

Figure 21 - Ranking de l'importance des caractéristiques de XGBClassifier

Excellent. La figure ci-dessus nous montre le classement des meilleurs caractéristiques et comme supposé lors de l'analyse de Figure 5 - Boîtes à moustaches entre attributs corrélés, ces trois caractéristiques pourraient suffire pour obtenir un très bon modèle.

b. Utilisation des caractéristiques *ejection_fraction*, *serum_creatinine* et *time*

On a vu que les trois caractéristiques *ejection_fraction, serum_creatinine* et *time* semblent être les plus importants pour notre tâche. Voyons si nous obtenons de bons résultats.

```
{'C': 1.0, 'class_weight': None, 'max_iter': 25, 'solver': 'lbfgs'}
{'class_weight': None, 'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 50}
{'eta': 0.2, 'gamma': 0, 'lambda': 50, 'max_depth': 2}
{'C': 0.1, 'class_weight': 'balanced', 'degree': 3, 'gamma': 'scale', 'kernel': 'poly'}
```

Figure 22 - Paramètres obtenus

On note les changements suivant :

- *Logistic Regression :*
  - *C : ~~0.01~~ -> 1*
  - *solver : ~~liblinear~~ -> lbfgs*
- *Random Forest* :
  - *criterion : ~~entropy~~ -> gini*
  - *n_estimators : ~~200~~ -> 50*
- *XGBClassifier :*
  - *gamma : ~~5~~ -> 0*
  - *lambda : ~~1~~ -> 50*
- SVC :

- *class_weight : ~~None~~ -> balanced*
- *kernel : ~~linear~~ -> poly*
- *degree : 3*
- *C : ~~1~~ -> 0.1*

Le changement de paramètres le plus flagrant est celui du *SVC* : le noyau a été changé et le modèle est désormais **pénalisé** grâce à *class_weights*.

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.555	0.645	0.817	0.833	0.526	0.739
RandomForestClassifier	0.641	0.727	0.850	0.857	0.632	0.791
XGBClassifier	0.603	0.667	0.833	0.909	0.526	0.751
SVC	0.511	0.600	0.800	0.818	0.474	0.712

Figure 23 - Résultats obtenus - Approche des trois meilleures caractéristiques

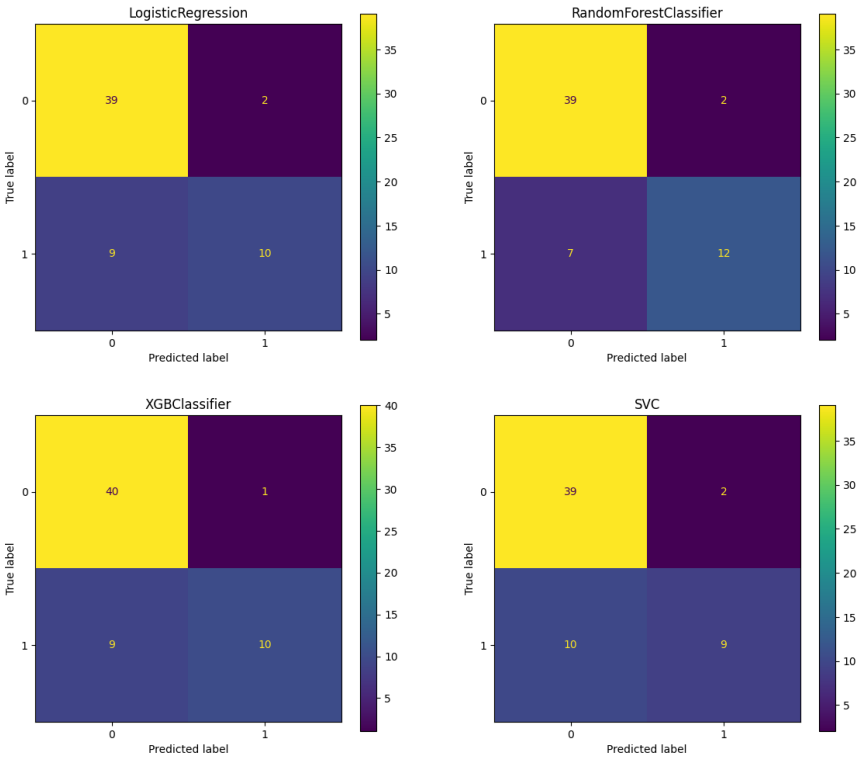

Figure 24 - Matrice de confusion - Approche des trois meilleures caractéristiques

Fascinant ! En utilisant uniquement trois caractéristiques on obtient les meilleurs résultats obtenus jusqu'à présent. Toutes les métriques des modèles ont été améliorées hormis pour la régression logistique.

Mais une chose est à souligner. Comme mentionné dans l'article, la caractéristique *time* (i.e. le temps de suivi du patient) n'est pas une caractéristique médicale au sens propre (ce temps de suivi ne dépend

pas de données mesurées directement sur le patient). Par conséquent, regardons comment se comportent les modèles sans cette caractéristique.

c. Utilisation des caractéristiques *ejection_fraction* et *serum_creatinine*

```
{'C': 0.1, 'class_weight': 'balanced', 'max_iter': 25, 'solver': 'lbfgs'}
{'class_weight': None, 'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 50}
{'eta': 0.4, 'gamma': 0, 'lambda': 50, 'max_depth': 2}
{'C': 0.1, 'class_weight': 'balanced', 'degree': 2, 'gamma': 'scale', 'kernel': 'linear'}
```

Figure 25 - Paramètres obtenus

On note ici quelques changement de paramètres mais moins flagrants que précédemment (~~*rbf*~~ *-> linear,* ~~*gini*~~ *-> entropy* et quelques autres).

Vu qu'on utilise deux caractéristiques, on en profite pour tracer la frontière de décision du *SVC*.

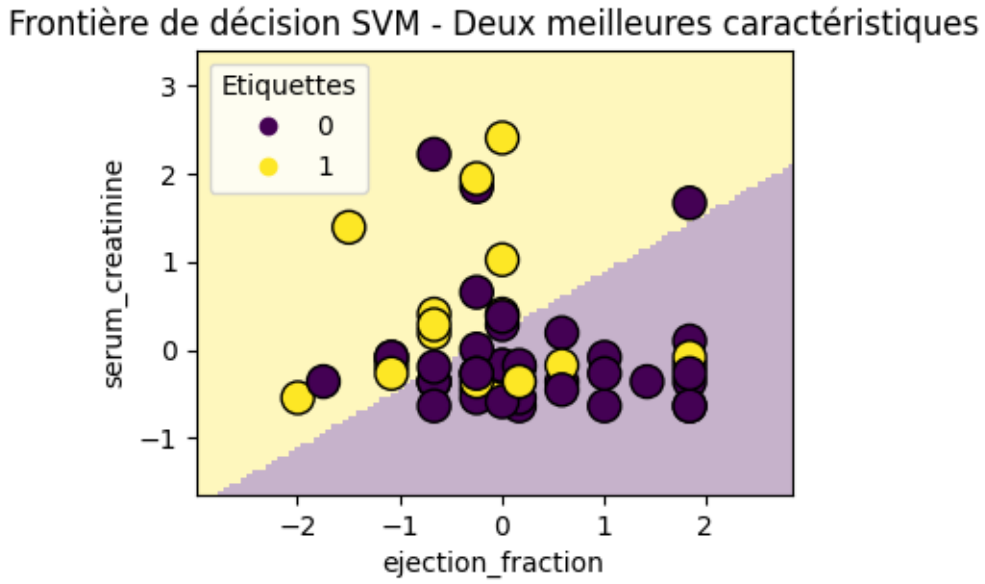

Figure 26 - Frontière de décision et nuage de points de l'ensemble de test

Aïe. On remarque que les deux caractéristiques et les étiquettes ne sont pas linéairement séparables du faite que certaines instances se confondent avec les instances de la classe opposée. On peut aussi noter la sur présence d'instances *y=0* du côté *y=1*.

La figure nous donne une indique que le SVC a du mal à effectuer les classifications. Regardons tout de même les métriques.

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.322	0.558	0.683	0.500	0.632	0.669
RandomForestClassifier	0.418	0.533	0.767	0.727	0.421	0.674
XGBClassifier	0.302	0.485	0.717	0.571	0.421	0.637
SVC	0.327	0.550	0.700	0.524	0.579	0.668

Figure 27 - Résultats obtenus

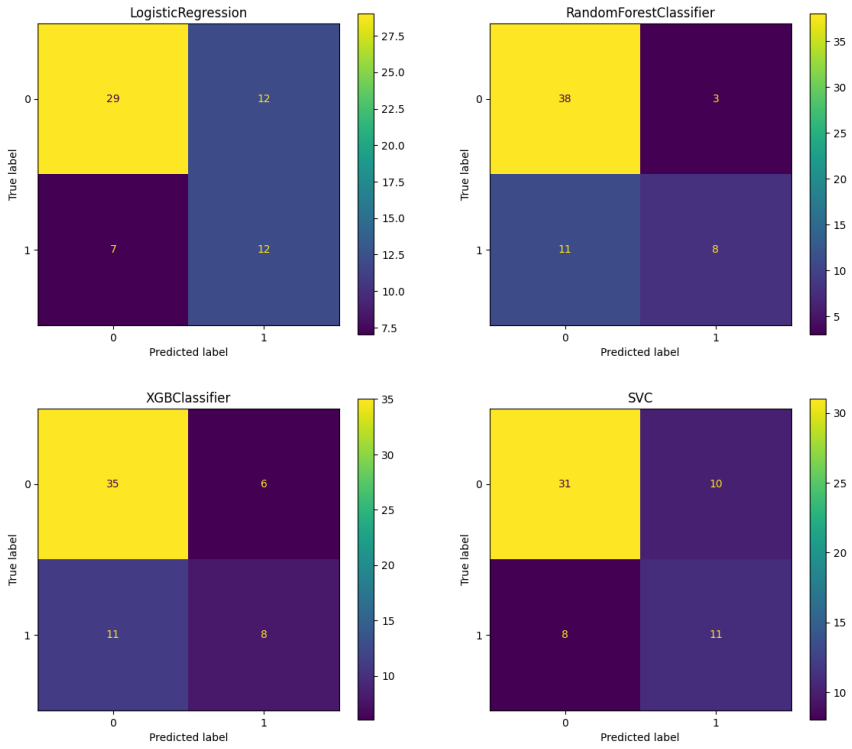

Figure 28 - Matrice de confusion - Approche des deux meilleures caractéristiques

Outre les mauvaises prédictions du *SVC* qu'on a remarqué, même avant la lecture des métriques, tous les scores ont baissé. Mais tout compte fait, lorsqu'on compare nos résultats avec Figure 9 - Résultats de l'étude en utilisant uniquement deux caractéristiques on se rends compte que l'étude obtiens plus ou moins les mêmes résultats (les MCC de notre *SVC* et de l'arbre boosté sont un peu plus bas).

On peut retenir quelques informations. Si on se focalise sur le MCC, l'unique utilisation des caractéristiques *ejection_fraction* et *serum_creatinine* nous montre une corrélation linéaire entre les observations et les prédictions. Cette métrique est intéressante car elle est **invariante au déséquilibre des classes**.

Les résultats sont considérés comme similaires à ceux de l'article de recherche.

d. Utilisation du sur-échantillonnage

Pour commencer, on divise notre jeu de données avant rééchantillonnage pour prévenir tout risque de sur-apprentissage.

```
{'C': 0.01, 'class_weight': 'balanced', 'max_iter': 25, 'solver': 'liblinear'}
{'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
{'eta': 0.2, 'gamma': 0, 'lambda': 10, 'max_depth': 6}
{'C': 10, 'class_weight': None, 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'}
```

Figure 29 - Paramètres obtenus

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.552	0.700	0.80	0.667	0.737	0.783
RandomForestClassifier	0.643	0.743	0.85	0.812	0.684	0.806
XGBClassifier	0.643	0.743	0.85	0.812	0.684	0.806
SVC	0.400	0.571	0.75	0.625	0.526	0.690

Figure 30 - Résultats obtenus

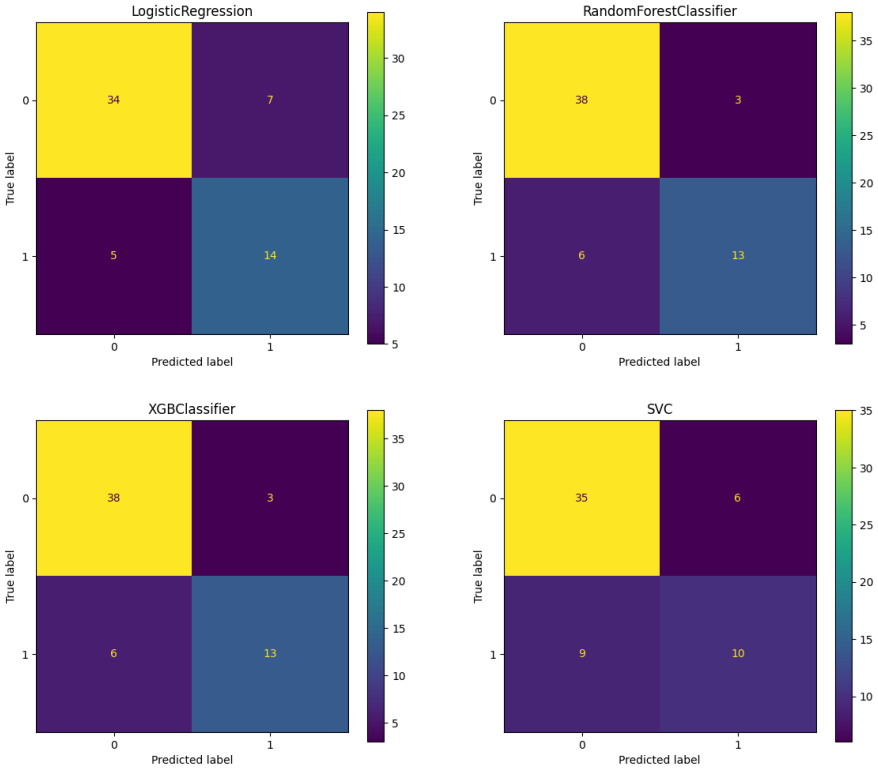

Figure 31 - Matrice de confusion - Approche sur-échantillonnage

L'approche du sur-échantillonnage montre de très bons résultats pour *RandomForest* et les résultats les plus élevés jusqu'à présent pour *XGBClassifier*. On retient cette solution qu'on combinera plus tard avec une autre approche pour voir si on peut encore améliorer nos résultats.

e. Utilisation du sous-échantillonnage

On effectue le même procédé que le sur-échantillonnage.

```
{'C': 1.0, 'class_weight': None, 'max_iter': 25, 'solver': 'lbfgs'}
{'class_weight': None, 'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 100}
{'eta': 0.4, 'gamma': 0, 'lambda': 50, 'max_depth': 4}
{'C': 10, 'class_weight': None, 'degree': 2, 'gamma': 'scale', 'kernel': 'linear'}
```

Figure 32 - Paramètres obtenus

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.447	0.611	0.767	0.647	0.579	0.716
RandomForestClassifier	0.538	0.684	0.800	0.684	0.684	0.769
XGBClassifier	0.606	0.722	0.833	0.765	0.684	0.793
SVC	0.384	0.579	0.733	0.579	0.579	0.692

Figure 33 - Résultats obtenus

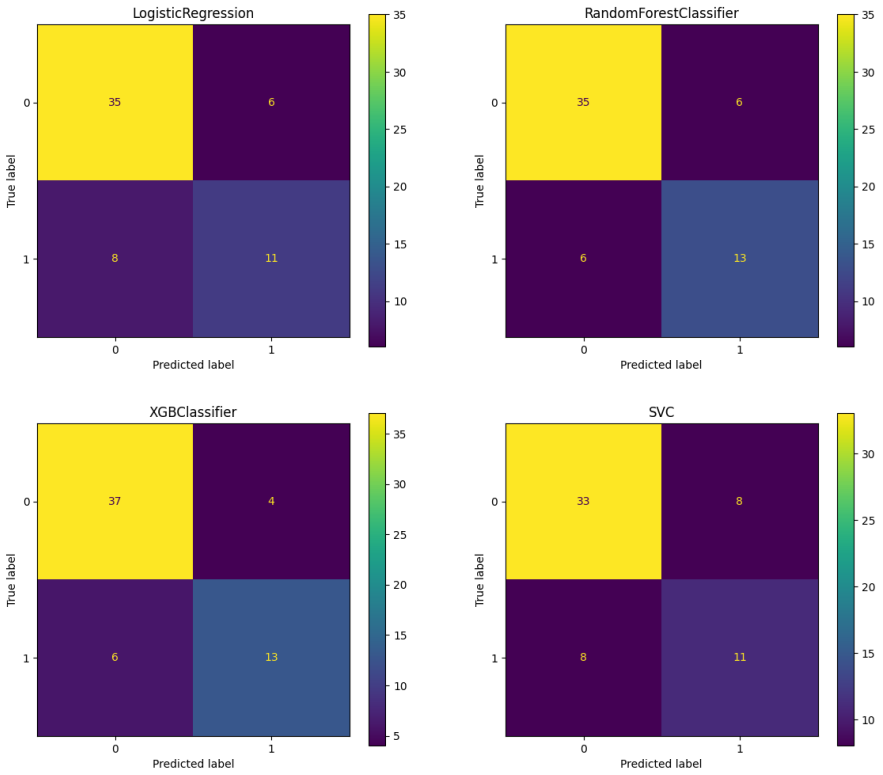

Figure 34 - Matrice de confusion - Approche sous-échantillonnage

L'approche du sous-échantillonnage montre des résultats décents pour *RandomForest* et *XGBClassifier* mais pas assez élevés pour qu'on puisse la considérer comme une approche intéressante.

f. Synthetic Minority Oversampling Technique (SMOTE)

```
{'C': 0.1, 'class_weight': None, 'max_iter': 25, 'solver': 'liblinear'}
{'class_weight': None, 'criterion': 'gini', 'max_depth': 25, 'min_samples_split': 2, 'n_estimators': 50}
{'eta': 0.8, 'gamma': 0, 'lambda': 0.1, 'max_depth': 4}
{'C': 10, 'class_weight': None, 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'}
```

Figure 35 - Paramètres obtenus

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.447	0.611	0.767	0.647	0.579	0.716
RandomForestClassifier	0.526	0.667	0.800	0.706	0.632	0.755
XGBClassifier	0.556	0.667	0.817	0.786	0.579	0.753
SVC	0.400	0.571	0.750	0.625	0.526	0.690

Figure 36 – Résultats obtenus

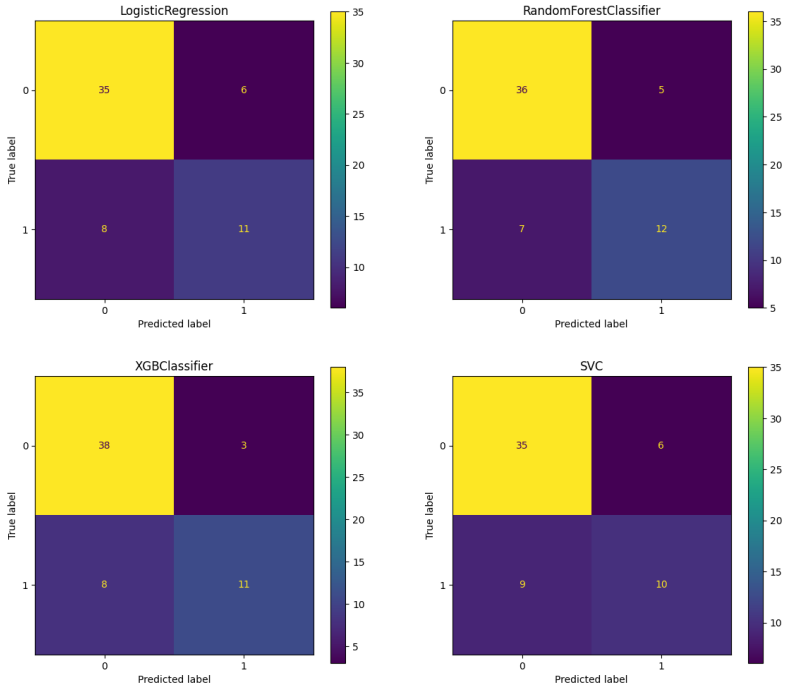

Figure 37 - Matrice de confusion - Approche SMOTE

La création d'exemples artificiels avec *SMOTE* ne montre pas non plus de résultats assez élevés pour qu'on puisse la considérer comme une approche intéressante.

3. Approche finale

Ce qu'on peut retenir de toutes les approches testées précédemment est que :

- Pour les stratégies de rééchantillonnage :

  Sur-échantillonnage > Sous-échantillonnage ≈ *SMOTE*

- Pour les approches basées sur les caractéristiques :

  serum_creatinine, ejection_fraction, time > all >> *serum_creatinine, ejection_fraction*

- Pour les approches basées mise à l'échelle et recherche d'hyperparamètres :
  - Sans *SVC :*

    Naïve > *Feature scaling* + *GridSearch*
  - Tout modèle confondu :

    Feature scaling + *GridSearch* > Naïve

Globalement, la stratégie de sur-échantillonnage est celle qui montre les meilleurs résultats.

Par conséquent, une bonne idée serait de combiner les trois meilleurs types d'approches pour voir ce qu'il en est. Notre tentative finale utilisera donc un rééchantillonnage combiné aux caractéristiques *ejection_fraction, serum_creatinine* et *time* avec une mise à l'échelle et une recherche d'hyperparamètres.

```
{'C': 10, 'class_weight': 'balanced', 'max_iter': 25, 'solver': 'lbfgs'}
{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
{'eta': 2, 'gamma': 0, 'lambda': 1, 'max_depth': 6}
{'C': 1, 'class_weight': None, 'degree': 2, 'gamma': 'auto', 'kernel': 'rbf'}
```

Figure 38 - Paramètres obtenus

	matthews_corrcoef	f1_score	accuracy_score	precision_score	recall_score	roc_auc_score
LogisticRegression	0.526	0.667	0.800	0.706	0.632	0.755
RandomForestClassifier	0.643	0.743	0.850	0.812	0.684	0.806
XGBClassifier	0.641	0.727	0.850	0.857	0.632	0.791
SVC	0.481	0.629	0.783	0.688	0.579	0.728

Figure 39 - Résultats obtenus

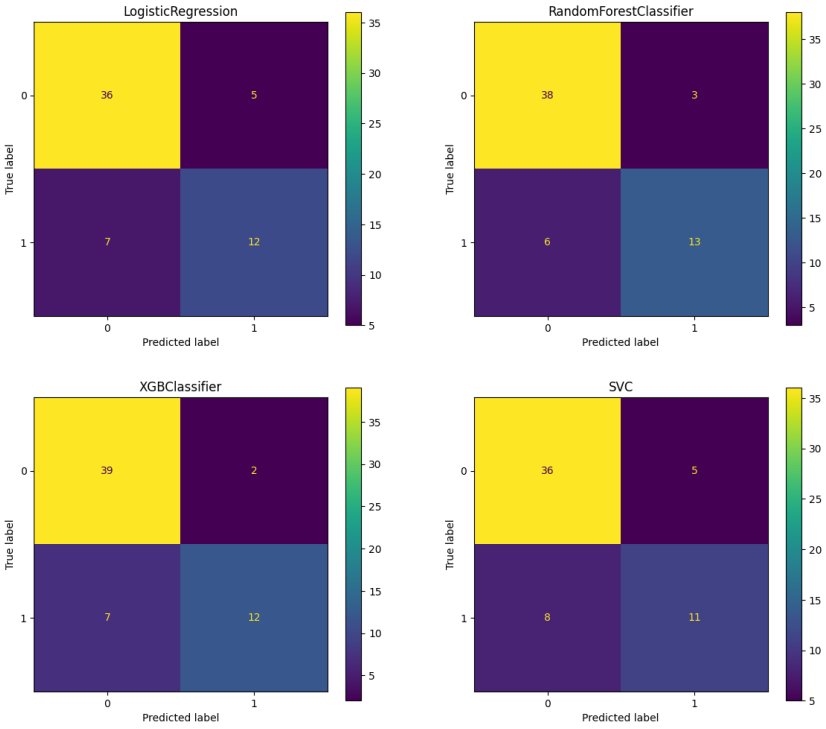

Figure 40 - Matrice de confusion - Approche finale

Dommage. Les résultats entre cette dernière approche et celle du sur-échantillonnage obtiennent les mêmes résultats, sauf pour le modèle SVM pour lequel les scores ont augmenté (mais qui restent malgré tout plus bas que les performances du SVM de la partie 2.b).

En sachant que nous n'utilisons que trois caractéristiques, que cela réduit les probabilités de sur-apprentissage de nos modèles et que l'entraînement serait nettement plus rapide si on possédait un jeu de données massif, cette solution est tout de même à retenir pour *RandomForest* et *XGBClassifier*.

Conclusion

L'apprentissage machine nous a permis d'obtenir de très bons résultats pour prédire la survie de patients atteints d'insuffisance cardiaque et ce de manière automatique. On a vu à travers différentes approches que les performances des modèles varient en fonction de celles-ci.

La régression logistique fonctionne le mieux lorsqu'aucune pré-traitement n'a été effectué (malheureusement on sait que l'algorithme d'optimisation de la régression logistique est sensible à la variation d'échelle et qu'avec suffisamment de recherche d'hyperparamètres, on obtiendrai un modèle qui se rapproche de celui de l'étude).

La forêt d'arbres décisionnels performe très bien lorsque nous utilisons uniquement les trois meilleures caractéristiques et l'ajout d'un sur-échantillonnage ne change pas ses performances.

L'arbre boosté excelle lorsque qu'un sur-échantillonnage est appliqué et l'ajout d'un sur-échantillonnage abaisse de peu ses performances.

La machine à vecteurs de support est dans ses meilleurs conditions lorsqu'on utilise uniquement les trois meilleures caractéristiques et ce, sans sur-échantillonnage.

Modèle	MCC	Exactitude
Régression Logistique – Article de recherche – Utilisation des trois meilleurs caractéristiques	0.616	0.838
Régression Logistique – Utilisation de toutes les caractéristiques (sans pré-traitement)	0.599	0.833
Forêt d'arbres – Utilisation des trois meilleurs caractéristiques	0.641	0.850
Arbre boosté – Utilisation de toutes les caractéristiques avec sur-échantillonnage	0.643	0.850
Machine à vecteurs de supports – Utilisation des trois meilleures caractéristiques	0.511	0.800

Les résultats obtenus tendent/dépassent (de peu) ceux de l'article de recherche. Nous noterons tout de même que leurs résultats sont, premièrement, mesurés sur une moyenne de cent exécutions et qu'ils sont par conséquent plus précis que les nôtres. Deuxièmement, que l'étude n'a effectué aucune stratégie de rééchantillonnage, contrairement à nous.

Nous nous sommes focalisés sur l'exactitude et le Coefficient de corrélation de Matthew, mais nous avons vu que dans un contexte médical réel, ce ne sont pas métriques les plus pertinentes.

On peut également citer les pistes non explorées lors du projet :

- Une recherche de d'hyperparamètres aléatoires – avec *RandomizedSearch* (surtout pour la régression logistique)
- Un ajout de différents types de régularisation lors de la recherche d'hyperparamètres
- Une mise à l'échelle normalisée (i.e. *MinMaxScaler*) – à noter que ce type de mise à l'échelle est sensible aux valeurs aberrantes
- Certains modèles:
    - Les k plus proches voisins (résultats épouvantables pendant l'étude mais on aurait tout de même pu vérifier par nous-mêmes)
    - La classification naïve bayésienne

Nous noterons finalement que le peu d'exemples de données (et leur manque de reproductibilité[3]) limite les performances de nos modèles et que l'utilisation des caractéristiques les plus pertinentes peuvent montrer de meilleurs résultats que l'utilisation de toutes les caractéristiques.

[3] Smith DH, Johnson ES, Thorp ML, Yang X, Petrik A, Platt RW, Crispell K. Predicting poor outcomes in heart failure. Permanente J. 2011;15(4):4–11.