Rapport - Défi IA 2020

M2 SID - UE EIMAB3CM - Apprentissage pour les données massives

Groupe randomSeed :

Emma Grandgirard

David Jeauneau

Christelle Latorre

Coraline Le Brun

Rendu le 18 mars 2020

Table des matières

1 Le défi et l'équipe

1.1 Présentation du défi

Le défi IA est une compétition de machine learning organisée chaque année par l'INSA de Toulouse et s'adressant à des étudiants en Master. Cette année, le but du défi était de détecter les anomalies sur des signaux provenant de capteurs d'hélicoptères en utilisant des méthodes de machine learning et/ou de deep learning. Les données étaient fournies par Airbus. Il s'agit d'un problème de classification binaire puisqu'il faut, pour chaque signal, déterminer s'il est anormal ou non. Le défi s'est déroulé en deux parties.

Pour la première partie (du 10 octobre 2019 au 1er janvier 2020), nous disposions d'un jeu de données d'apprentissage qui ne contenait que des signaux sans anomalies, et d'un jeu de validation qui contenait des signaux avec et sans anomalies. Le but était d'entraîner des modèles sur le jeu d'apprentissage, afin qu'ils apprennent les caractéristiques des signaux normaux. Une fois appliqués au jeu de validation, ces modèles devaient parvenir à distinguer les signaux sans anomalies des signaux avec anomalies. Il s'agissait donc d'un problème d'apprentissage semi-supervisé. Nous avions droit à un nombre illimité de soumissions. Pour chaque soumission, un score (F1-score avec précision et rappel, voir Partie 2.2) était donné au bout d'une heure. Nous pouvions également consulter le classement des équipes (établi selon le meilleur F1-score de chaque équipe).

Pour la seconde partie (du 2 au 15 janvier 2020), nous devions détecter les anomalies dans un nouveau jeu de données, le jeu de test. Nous ne pouvions proposer que deux soumissions, sans connaître leur score avant l'annonce des résultats (23 janvier 2020), d'où l'importance du choix des modèles et de la méthode de validation.

1.2 Présentation de l'équipe et organisation

Notre équipe randomSeed était composée de 4 personnes : Emma Grandgirard, David Jeauneau, Christelle Latorre et Coraline Le Brun.

Tout au long du défi, nous avons utilisé un répertoire GitHub privé pour partager notre code (en Python). Chacun créait ses propres features et modèles qui étaient mis en commun sur GitHub. De cette manière, chaque membre du groupe pouvait réutiliser facilement les codes des autres. Nous avons également créé un fichier Google Sheets où étaient nous tenions à jour la liste des soumissions effectuées. Pour chaque soumission étaient indiquées les informations suivantes : nom de la soumission, date et heure, F1-score, précision, rappel, ainsi que les features et modèles utilisés. Un code couleur nous permettait rapidement de situer chaque score par rapport aux scores des autres soumissions.

2 Données, objectifs et stratégie

2.1 Les données

Les données étaient des signaux provenant de capteurs d'hélicoptères, fournies par Airbus au format hdf5. Il s'agissait de séquences d'une minute de mesure d'accéléromètre à une fréquence de 1 024 Hertz, donnant une longueur égale à 61 440. Ces mesures ont été effectuées sur des hélicoptères lors de différents vols. Nous n'avions pas d'information sur le moment du vol, la localisation de l'appareil et les angles des capteurs au moment où les mesures ont été effectuées. D'après les informations fournies par les organisateurs, les données avaient été multipliées par un facteur, mais aucune autre normalisation n'avait été faite. De plus, si une anomalie était détectée à moment donné dans le vol, tous les enregistrements de ce vol étaient classifiés comme anormaux, même ceux précédant le moment où l'anomalie était constatée.

L'image ci-dessous représente le profil d'un signal, avec l'affichage de fenêtres représentant des zooms forts sur certaines parties du signal permettant de mettre en évidence sa nature (cyclique ou aléatoire).

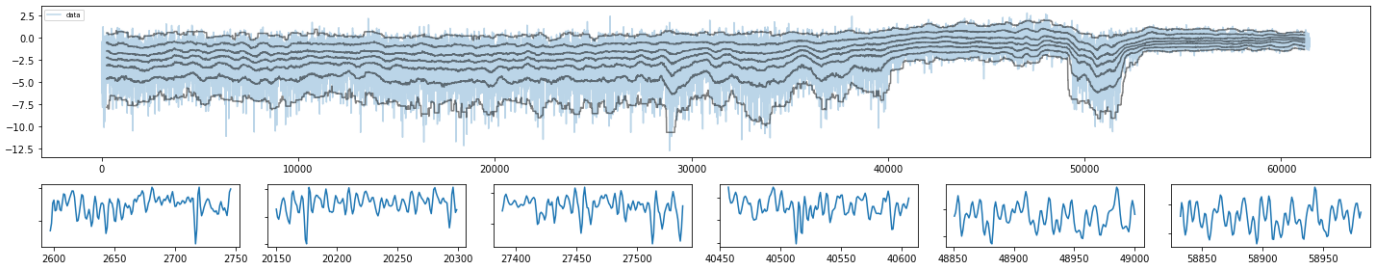

Fig. 1 : Visualisation du profil d'un signal

Le jeu d'apprentissage contenait 1677 séquences, le jeu de validation 594 séquences, et le jeu de test 1917 séquences. Le jeu de validation était composé à 50% d'anomalies, en revanche nous ne connaissions pas la proportion d'anomalies dans le jeu de test.

2.2 Objectifs quantitatifs du défi

La métrique choisie par les organisateurs pour évaluer les soumissions était le F_β-score, calculé à partir de deux métriques statistiques : la précision et le rappel. La précision est le nombre d'anomalies correctement identifiées (vrais positifs) divisé par le nombre d'anomalies prédites. Le rappel est le nombre d'anomalies correctement identifiées (vrais positifs) divisé par le nombre d'anomalies présentes dans le jeu de données. Le F_β-score est donné par la formule :

$$F_\beta = \frac{(1+\beta^2) \times Précision \times Rappel}{\beta^2 \times Précision + Rappel}$$

avec β un élément à déterminer pour donner davantage de poids à l'une ou l'autre des métriques. Dans ce défi, β a été fixé à 1 afin de donner à la précision et au rappel une importance égale. Le score calculé était donc le F_1-score, qui correspond à la moyenne harmonique du rappel et de la précision, donnée par la formule :

$$F_1 = \frac{2 \times Précision \times Rappel}{Précision + Rappel}$$

L'objectif était d'obtenir le meilleur F_1-score possible, en particulier lors de la seconde partie du défi.

2.3 Stratégie globale

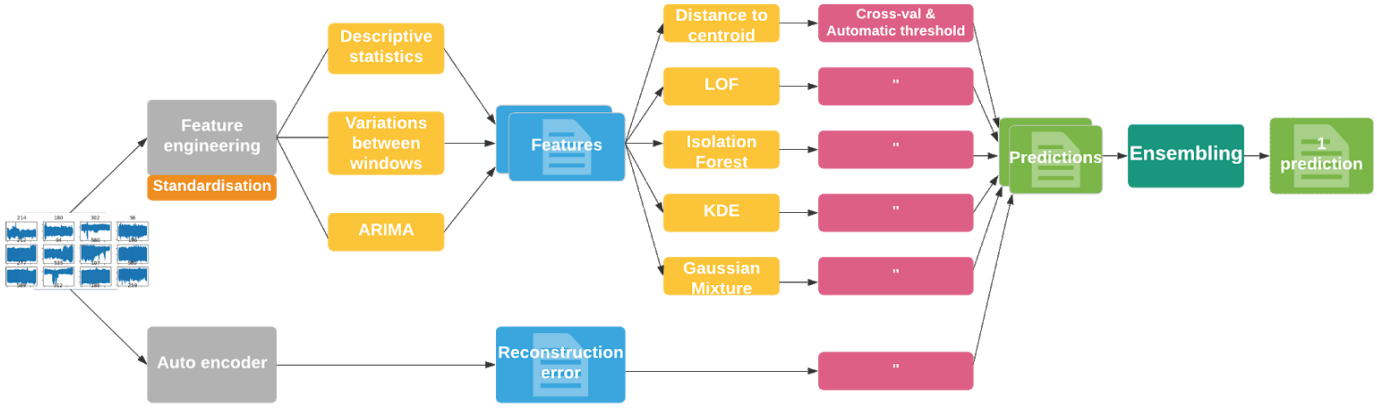

Fig. 2 : Pipeline de traitement global

Le schéma ci-dessus résume les traitements que nous avons réalisés pour répondre au problème, et que nous présenterons dans les parties 3 et 4 de ce rapport, la partie 5 étant consacrée aux résultats et aux choix stratégiques pour la seconde phase.

La moitié haute du schéma correspond à l'approche machine learning, et la moitié basse l'approche deep learning.

Pour réduire la dimensionnalité des jeux de données (partie 3), nous avons d'une part procédé à la création de features (3.1), et d'autre part construit un auto-encodeur (3.2). Pour chaque signal, le résultat était un ensemble de 20 features côté machine learning, et une erreur de reconstruction côté deep learning. Nous pensons que le fait d'avoir mis en place des méthodes variées de création de features est l'un des points forts de notre stratégie : grâce aux statistiques descriptives, aux calculs de différences entre fenêtres et à l'utilisation du modèle ARIMA, nous avons trouvé des moyens d'identifier plusieurs types de différences entre les signaux normaux et anormaux.

La partie droite du schéma concerne la détection d'anomalies (partie 4). Dans la partie machine learning, nous avons appliqué différents modèles connus (LOF, Isolation Forest, KDE, Gaussian Mixture), ainsi qu'un modèle basé sur un calcul de distance euclidienne, pour donner un score d'anomalie à chaque signal (4.1). Ensuite, nous avons mis en place une méthode basée sur la cross-validation pour trouver le meilleur seuil de manière quasi automatique pour chaque liste de

scores (4.2). Du côté deep learning, l'erreur de reconstruction des signaux par l'auto-encodeur constituait le score d'anomalie ; nous avons également appliqué ici une méthode de cross-validation pour déterminer le seuil.

Chaque méthode donnait comme résultat une liste de prédictions sous forme de classification binaire (anomalie ou non). Lors de la première phase du défi, nous avons fait des soumissions correspondant à différentes combinaisons features / modèle, puis nous avons mis en place des méthodes d'ensemble par vote (4.3) qui nous ont permis d'améliorer nos scores.

3 Pré-processing

3.1 Création de features

3.1.1 Statistiques descriptives

Pour réduire la dimensionnalité des données, nous avons, dans un premier temps, décidé d'utiliser des statistiques descriptives simples comme la moyenne, la médiane, l'écart interquartile ou encore la variance. Nous avons calculé ces statistiques sur tout le signal mais aussi sur des fenêtres glissantes du signal. Le faire sur des fenêtres glissantes permet de mieux capter les changements qu'il y a eu tout au long du signal. C'est notamment utile pour connaître les signaux qui fluctuent beaucoup au cours du temps et qui peuvent donc potentiellement être des anomalies.

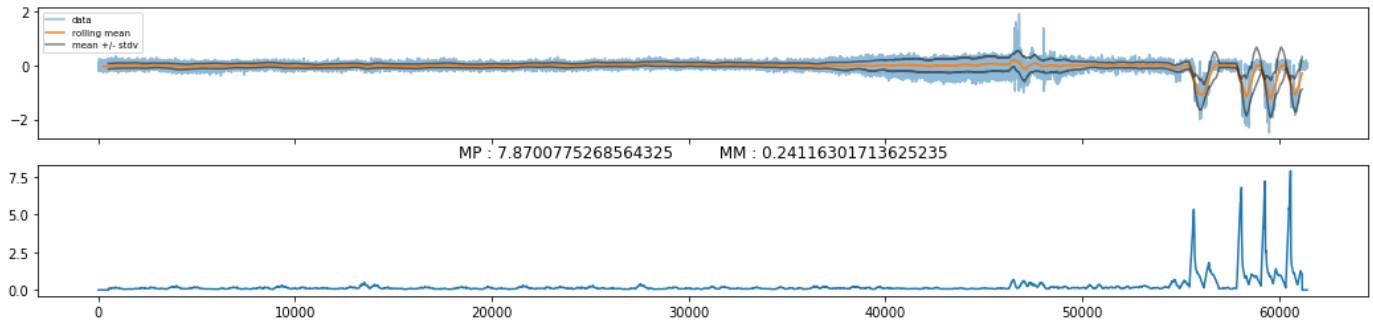

Fig. 3 : Visualisation de la création des features MP et MM sur un signal

Par exemple (figure ci-dessus), la feature MP essaye de capturer la variation la plus violente dans un signal en comparant la fluctuation de la moyenne d'une fenêtre à l'autre, et en normalisant par la l'écart-type moyen sur ces fenêtres. Dans cet exemple, on observe que la moyenne a au maximum dévié de MP = 7.87 écarts-types à un moment de ce signal. MM correspond quant à elle à la fluctuation moyenne, en écart-type, sur l'ensemble du signal.

Nous avons également créé une feature égale à la valeur absolue de la différence entre le nombre de valeurs du signal au-dessus de la moyenne du signal et le nombre de valeurs en-dessous de la moyenne (range_above_under_mean). Nous avons supposé que les valeurs des signaux anormaux ne devaient pas se répartir de façon équilibrée autour de la moyenne.
Nous avons également calculé l'entropie de chaque signal, qui est une mesure permettant de mesurer la complexité des différents signaux.

De plus, étant donné qu'il était indiqué sur le site de l'INSA que pour chaque signal les valeurs avaient été multipliées par un coefficient, nous avons essayé de faire un travail de standardisation au moment de la création de nos features afin de ne pas avoir à prendre en compte la valeur absolue des données. Nous avons alors créé la feature nommée EB (Estimated Bias) qui est égale au quotient de la moyenne par l'écart-type (égal à 0 si l'écart-type est égal à 0). Cependant, si cette feature s'est avérée très intéressante pour obtenir un bon score dans la première phase, nous nous sommes aperçus lorsque nous avons reçu le jeu de test que les nouvelles anomalies avaient cette fois-ci des valeurs similaires aux données normales.

Nous avons également utilisé la librairie tsfresh, qui permet de calculer un grand nombre de caractéristiques des séries temporelles, puis, en fonction des résultats obtenus dans nos soumissions, nous avons gardé les features qui nous donnaient les meilleurs résultats :

- abs_energy : l'énergie absolue de la série, soit la somme des carrés de ses valeurs ;
- longest_strike_above_mean : la taille de la plus longue séquence de valeurs au-dessus de la moyenne de la série ;
- c3 : coefficient mesurant la non-linéarité de la série en fonction d'une longueur de fenêtre, donné par :

$$\frac{1}{n-2lag}\sum_{i=0}^{n-2lag} x_{i+2\cdot lag}^2 \cdot x_{i+lag} \cdot x_i$$

avec n la longueur de la série, et lag la longueur de fenêtre choisie.

3.1.2 Modèle ARIMA

Autoregressive Integrated Moving Average (ARIMA) est une méthode statistique de prédiction de séries temporelles. Nous l'avons utilisée pour mesurer l'imprédictibilité de chaque signal, avec l'intuition qu'une série avec un haut score était probablement une anomalie. Nous avons d'abord divisé chaque signal en fenêtres de longueur 120, avant de calculer l'énergie absolue de chacune (la somme de ses valeurs au carré) et de retenir 3 fenêtres sur lesquelles appliquer le modèle ARIMA : les fenêtres d'énergie absolue maximale, médiane et minimale. Pour chacune de ces 3 fenêtres, nous avons entraîné le modèle ARIMA sur la première moitié puis prédit la seconde moitié, avant de calculer la RMSE entre la prédiction et la séquence originale. Les 3 RMSE obtenues constituaient une feature chacune. En retenant ces 3 fenêtres pour l'analyse, nous souhaitions voir si les anomalies étaient plus facilement repérables dans des zones de haute, moyenne ou basse énergie. En testant ces features lors de la première phase, nous avons pu remarquer que seule celle basée sur la fenêtre d'énergie maximale nous permettait d'améliorer nos scores.

La figure ci-dessous présente deux fenêtres avec signal original et prédiction sur la seconde moitié. A gauche, la fenêtre provient d'un signal du jeu d'entraînement, et celle de droite provient d'un signal du jeu de validation classé comme anomalie par tous nos modèles.

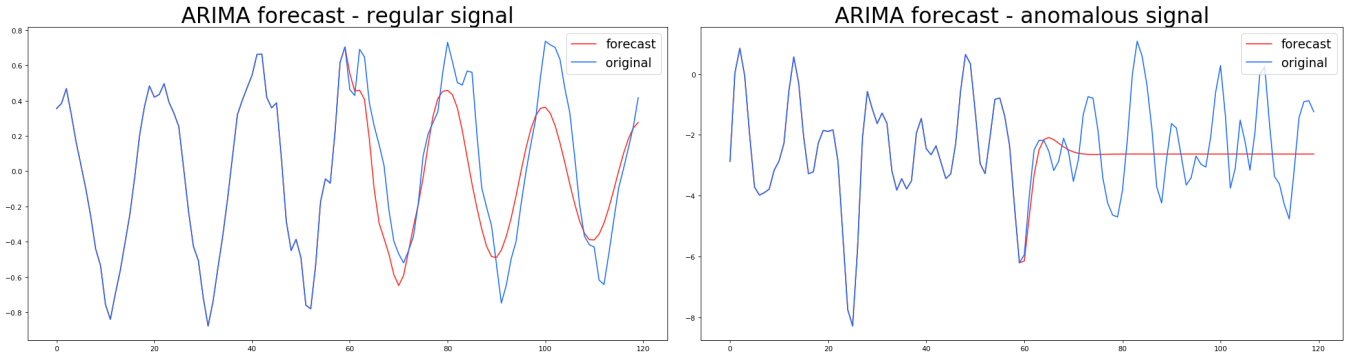

Fig. 4 : Méthode ARIMA

3.1.3 Utilisation de nos features

Toutes les features calculées précédemment nous ont permis de réduire la dimensionnalité de nos données. Pour chaque signal, nous sommes passés de 61 440 valeurs à 20 valeurs.

Nous avons par ailleurs mis l'accent sur la visualisation des données, des features et des modèles. Nous avons visualisé la distribution nos features une par une (boxplot), deux par deux (scatterplot) ou trois pas trois, comme présenté dans l'exemple ci-dessous, afin d'essayer de repérer visuellement les features qui permettaient d'isoler certains signaux des jeux de validation et de test, et de voir quelles paires ou trios de features pouvaient être utilisées ensemble pour faciliter cette séparation.

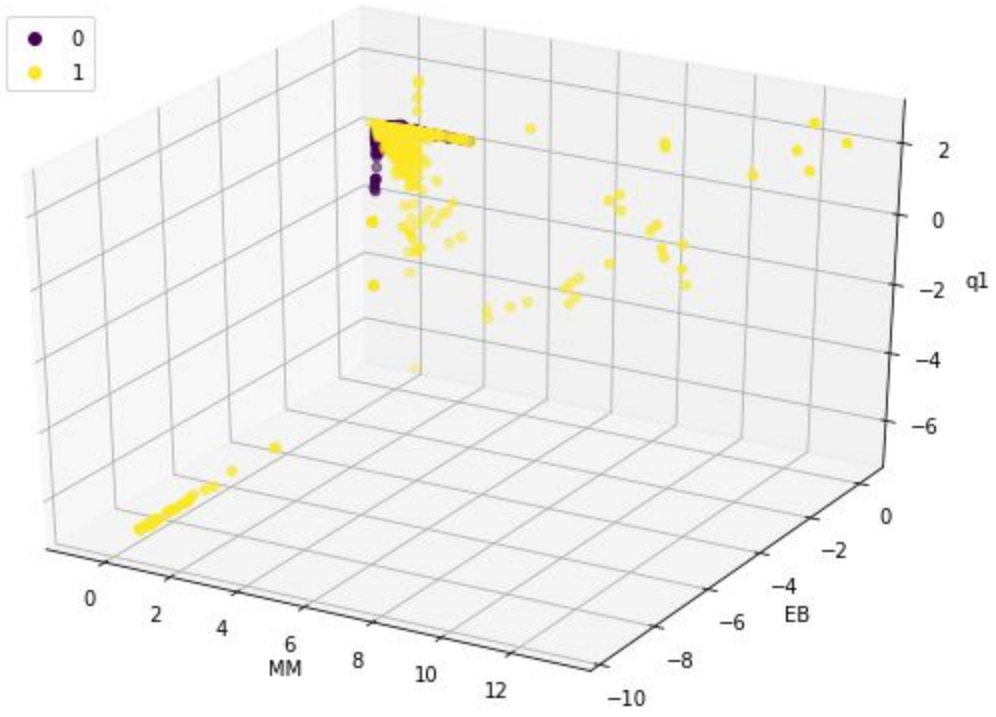

Fig. 5 : Visualisation de 3 features

Nous pouvons observer ci-dessus la représentation graphique de 3 features : MM (fluctuation moyenne entre fenêtres, en écart-type, sur l'ensemble du signal), EB (quotient de la moyenne par l'écart-type, égal à 0 si l'écart-type est égal à 0), et q1 (quantile 0.1), avec en violet les points du jeu

d'entraînement et en jaune ceux du jeu de validation. Nous remarquons que ces trois features permettent d'isoler un certain nombre de points du jeu de validation.

Nous avons également implémenté un paralléliseur afin d'optimiser les temps de calcul, et une "banque de features" pour stocker les résultats sous forme optimale. Le fonctionnement est le suivant : un ensemble de fonctions {f, …}, qui pour chaque signal calculent les features A et B pour f… est donné en paramètre au paralléliseur qui répartit les calculs sur n processeurs avant de recombiner les résultats. Ces résultats sont stockés dans autant de dataframes que de fonctions : le premier porte le nom de la fonction f et est composé des deux colonnes : A et B.

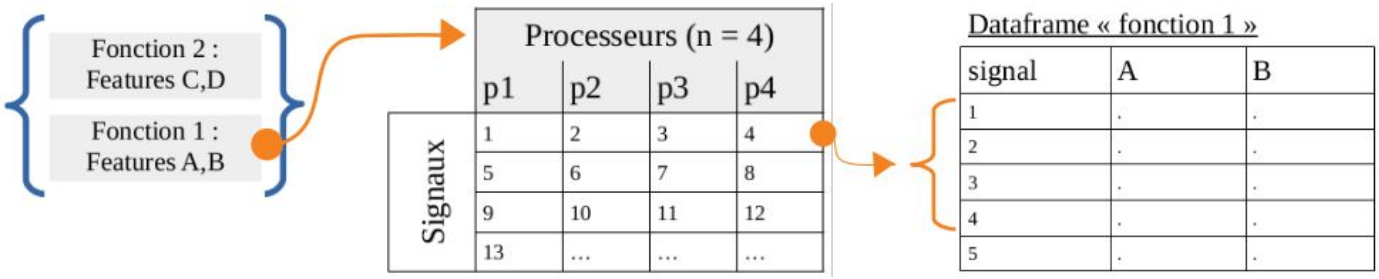

Fig. 6 : Fonctionnement du paralléliseur et de la banque de features

3.2 Auto-encodeur

Une autre approche de réduction de dimensionnalité que nous avons utilisée est l'auto-encodeur. L'auto-encodeur est un réseau de neurones qui permet de compresser les données dans un espace plus petit que l'espace de départ. Le réseau de neurones apprend les caractéristiques du signal normal qui lui permettent d'être compressé puis décompresse le signal en essayant de reconstruire au mieux le signal de départ. Si la reconstruction du signal est proche du signal de départ, alors il est probable que ce soit un signal sans anomalies. Au contraire, s'il y a un taux d'erreur élevé entre le signal obtenu avec le réseau de neurones et le signal de départ, alors il est probable que ce soit un signal avec anomalies.

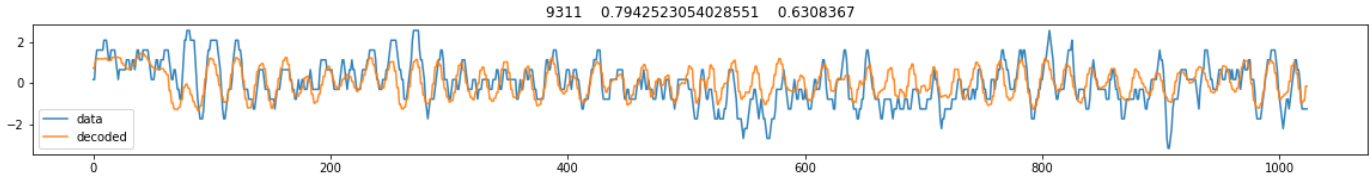

Fig. 7 : Signal original normalisé (bleu) et reconstruit par l'auto-encoder (jaune)

Nous avons testé plusieurs approches et plusieurs architectures de réseau de neurones et avons retenu la suivante :

- découpage de chaque signal par fenêtre coulissante de 1 seconde, avec un pas de 0.5 seconde (chaque signal de 1 minute est transformé en 123 signaux de 1 secondes) ;
- standardisation de ces signaux découpés (moyenne à 0, écart type à 1) ;
- choix de la fonction de perte : huber loss ou mean squared error ou mean absolute error ;
- architecture profonde basée sur ResNext (4 modules pour encoder, 4 pour décoder) ;
- dropout aléatoire, dropout de features et batchNormalization ;
- optimiseur : Adam ;
- activation : ReLu ;
- nombre de paramètres à entraîner : 63 946 ;
- compression : 16x.

Nous avons choisi de découper les signaux pour faciliter leur standardisation et ainsi éviter de donner trop d'importance à ceux évoluant dans des valeurs absolues élevées, ce qui aurait pénalisé l'apprentissage puisque l'algorithme n'aurait été focalisé que sur les grandes erreurs obtenues avec ces derniers signaux. Pour cela, nous avons utilisé une fenêtre coulissante d'une longueur de 1 seconde, avec un recouvrement de moitié puisque nous avions utilisé un padding "same" dans les convolutions, ce qui tend à détériorer la qualité du signal sur les bords. Après découpage, chaque fenêtre était donc standardisée en soustrayant la moyenne et en divisant par l'écart type. C'est par ailleurs une opération réversible lorsqu'on stocke ces informations, ce qui nous a ainsi permis de reconstruire les signaux jusqu'aux plus près de leurs versions originales et de ne pas rester avec leurs versions normalisées.

Nous avons également testé plusieurs fonctions de perte : mean absolute error, mean squared error, et un compromis entre les deux : la huber-loss avec un alpha de 1.5. Nous avons voulu voir l'influence potentielle sur l'apprentissage, en focalisant plus ou moins l'algorithme sur les données aberrantes, mais cela n'a pas eu d'effets notables ici.

Une architecture de type ResNext a été utilisée (voir Annexe), et beaucoup ont été testées. Nous voulions pouvoir compresser un signal 16 fois, en le divisant par deux 4 fois successivement. Pour cela nous avons utilisé deux fois 4 modules de type inception : 4 pour la compression, et 4 pour la décompression. Avec les réseaux de type inception de première génération, l'apprentissage ne débutait pas car le gradient ne pouvait pas se propager à traver le réseau. C'est pourquoi nous avons ajouté des connexions en court-circuit à la façon des réseaux de neurones résiduels pour ainsi obtenir une architecture de type ResNext et permettre au réseau de ne pas bloquer le gradient.

Afin de limiter le nombre de paramètres à entraîner, aucune couche entièrement connectée n'a été utilisée. Ce faisant nous avions finalement à entraîner seulement 63 946 paramètres, malgré la profondeur du réseau. Ce chiffre est à mettre en rapport avec d'un côté le nombre de données d'entraînement qui était de 187 663 (données découpées et standardisées), et d'un autre avec la taille de chacune : 1 024 (1 seconde) ; ou encore avec la taille d'un signal original : 61 440.

Pour éviter l'over-fit sur le jeu d'entraînement et obtenir de bonnes couches de features tout au long du réseau, nous avons utilisé les méthodes de batchNormalization et de dropout de features. Cela permettait pour la première méthode de pouvoir bien entraîner tous les niveaux, et pour la seconde de maximiser les différences entre les features d'un même niveau. Nous avons aussi essayé avec un dropout aléatoire afin d'éviter que chaque feature soit trop sensible, mais cela finissait par trop pénaliser l'entraînement.

Toutes nos fonctions d'activation étaient de type ReLu et nous avons utilisé l'optimiseur Adam de manière standard. 100 signaux sur les 1677 du jeu d'entraînement ont aussi été choisis aléatoirement pour être conservés pour la validation.

Finalement, plusieurs choix étaient possibles pour ensuite associer à chaque signal d'origine (de 1 minute) son score : la moyenne (ou la somme) de toutes ses erreurs sur toutes ses sous-parties, ou le maximum, ou une combinaison des deux. Nous avons finalement opté pour la somme, qui nous

a donné les meilleurs résultats. Etant donné que l'auto-encodeur est un modèle "à part", nous avons préféré utiliser son résultat (la somme des erreurs) seul plutôt que de l'intégrer comme feature dans un de nos modèles de machine learning. Nous avons obtenu comme meilleur score :

model	F1-score	Precision	Recall
Auto-encoder	**0,90716**	**0,92256**	**0,89226**

4 Détection des anomalies

Dans cette partie, nous allons présenter les modèles de machine learning qui nous ont permis de calculer des scores d'anomalie à partir des features présentées en partie 3.1. Nous expliquerons également notre méthode pour déterminer le seuil optimal, basée sur la cross-validation, ainsi que la méthode d'ensemble mise en oeuvre pour détecter un maximum de types d'anomalies et généraliser au mieux.

Avant d'utiliser certaines méthodes, dans un premier temps certaines anomalies évidentes étaient directement jugées comme telles et retirées du jeu de test. Pour les mettre en évidence, le jeu de test et de train étaient dans un premier temps fusionnés, avant qu'on standardise leurs features, et qu'on retire du jeu les données aberrantes, qui appartenaient toujours au jeu de validation / test.

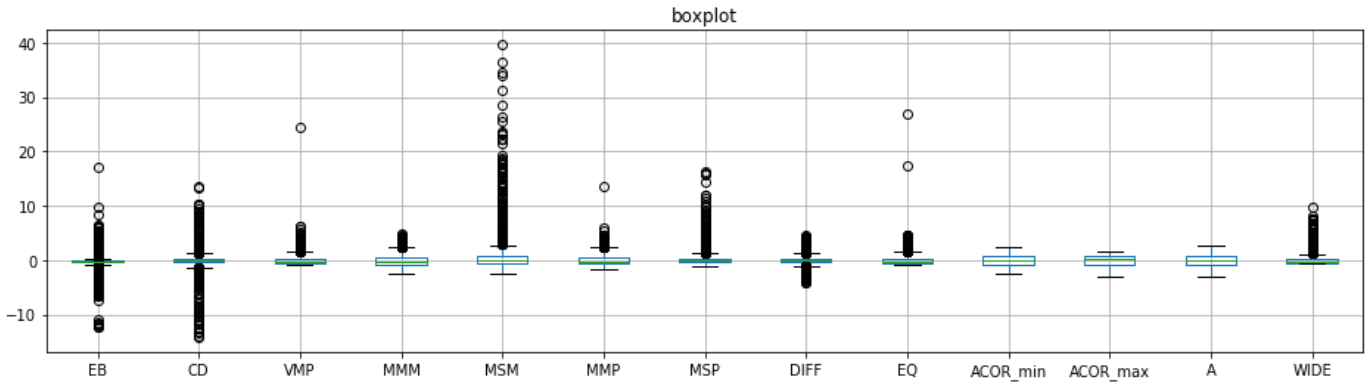

Fig. 8 : Standardisation des features - boxplots

Ces données aberrantes étaient par exemple les signaux dont le coefficient directeur médian (sur l'ensemble des fenêtres de 1 seconde) était bien trop éloigné de 0 (feature CD) indiquant une forte pente générale, ou encore les données dont la feature EB (estimation du biais) indiquait qu'on avait affaire à des données extrêmement décentrées.

Par ailleurs, un espace de haute dimensionnalité doit être en général évité avec certaines méthodes basées sur des calculs de distances. C'est pourquoi toutes les features intéressantes n'étaient pas

toujours utilisées à la fois avec tous les modèles. Une sélection des plus discriminantes était donc mise en oeuvre lorsque c'était nécessaire. Cela nous permettait aussi de rester en contact avec nos données, et de pouvoir comprendre et expliquer certains comportements.

4.1 Méthodes de détection des anomalies

Nous avons choisi plusieurs méthodes de machine learning différentes, basées sur des distances, des densités locales (Local Outlier Factor) et globales (Gaussian Mixture Model et Kernel Density Estimator), et basées sur des arbres (Isolation Forest), afin d'essayer de capturer au mieux les caractéristiques des différents signaux, et en essayant de cumuler les différents avantages de ces méthodes.
Toutes ces méthodes ont été optimisées individuellement dans un premier temps, en essayant de fixer leurs hyperparamètres, avant d'être combinées pour effectuer nos prédictions.

4.1.1 Distance au barycentre

Il nous a semblé intéressant, en guise de première approche, de calculer les statistiques descriptives usuelles et de pouvoir les visualiser facilement. Nous avons décidé d'en utiliser quelques unes dans un modèle relativement simple et intuitif, basé sur la distance au barycentre.

Pour essayer de discriminer visuellement les données normales des anomalies, nous avons placé chacun des enregistrements sur un graphique où la moyenne apparaissait en abscisse et l'écart-type en ordonnée (voir fig. 9 et 10 page suivante). Il est à noter que nous avons choisi l'écart-type plutôt que la variance car la moyenne et l'écart-type sont exprimés dans la même unité.
Lors de la phase 1, les données du jeu d'entraînement et celles du jeu de validation donnaient des graphiques différents. Nous avons donc considéré qu'il s'agissait d'une piste intéressante pour l'utilisation de ces features.

On peut notamment remarquer que contrairement aux deux premiers jeux de données, il ne semble pas y avoir de signaux dont la moyenne des valeurs est négative dans le jeu de test. De plus la moyenne de validation est négative et inférieure à celle du jeu d'entraînement, alors que la moyenne du jeu de test est positive.

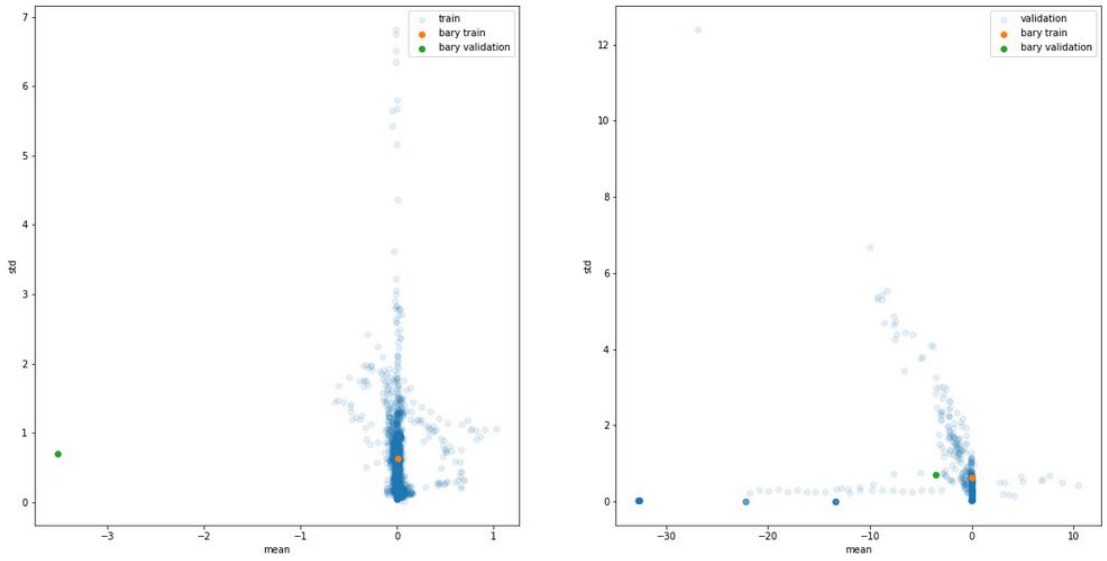

Fig. 9 : Représentation graphique des moyennes et écarts-types - Phase 1

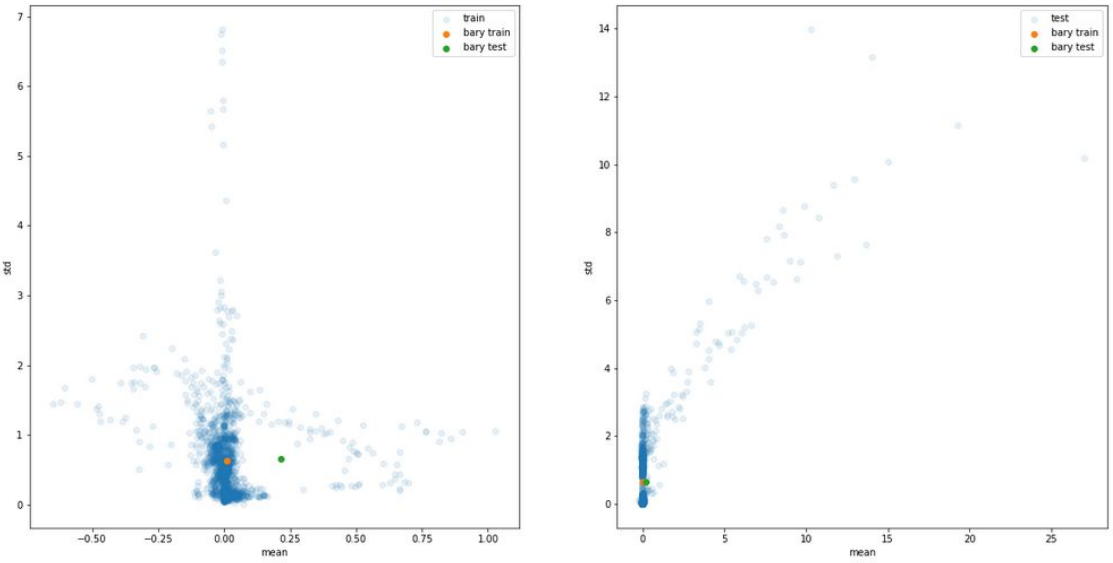

Fig. 10 : Représentation graphique des moyennes et écarts-types - Phase 2

Lors de la première phase, observer un barycentre du jeu de validation très éloigné des observations du jeu d'entraînement nous a poussé à mettre en place une méthode basée sur les distances au barycentre en utilisant ces features.

En utilisant la moyenne et l'écart-type, nous avons calculé le barycentre du jeu d'entraînement. Pour le jeu de validation, nous avons calculé la distance euclidienne entre chaque point et le barycentre du jeu d'entraînement. Un point était classé comme anomalie si sa distance au barycentre était supérieure à un seuil correspondant à 50% d'anomalies dans le jeu de validation. Nous avons également utilisé la feature range_above_under_mean, à laquelle nous avons appliqué une méthode similaire pour déterminer si la valeur était normale (proche de la moyenne pour le jeu d'entraînement) ou anormale. En assemblant ces deux classifications, nous avons pu classer comme anomalies les points trop éloignés des valeurs moyennes du jeu d'entraînement selon les 3 features citées. Pour la seconde phase, nous avons procédé de la même manière sur le jeu de test, en utilisant le seuil trouvé lors de la phase 1. Nous avions toutefois l'intuition que ce modèle serait moins performant sur le jeu de test, la visualisation graphique étant beaucoup moins concluante (fig. 10). Nous avons considéré que nous pouvions nous permettre de le conserver pour la seconde phase, mais uniquement en l'intégrant dans une méthode d'ensemble (4.3).

Le meilleur score obtenu lors de la phase 1 avec ce modèle est :

model	features	F1-score	Precision	Recall
Distance to centroid	mean, std, range_above_under_	**0,94416**	**0,94898**	**0,93939**

4.1.2 Local Outlier Factor

Le Local Outlier Factor (LOF) est une méthode basée sur la comparaison de la densité d'un point avec celle de ses k voisins. Si le score d'un point (un signal) est nettement différent de 1, cela signifie que la densité du point est très différente de celle de ses voisins, et que ce point peut être par conséquent classifié comme anomalie.

Le nombre de voisins à utiliser dans ce modèle à été fixé à 20 après quelques essais.

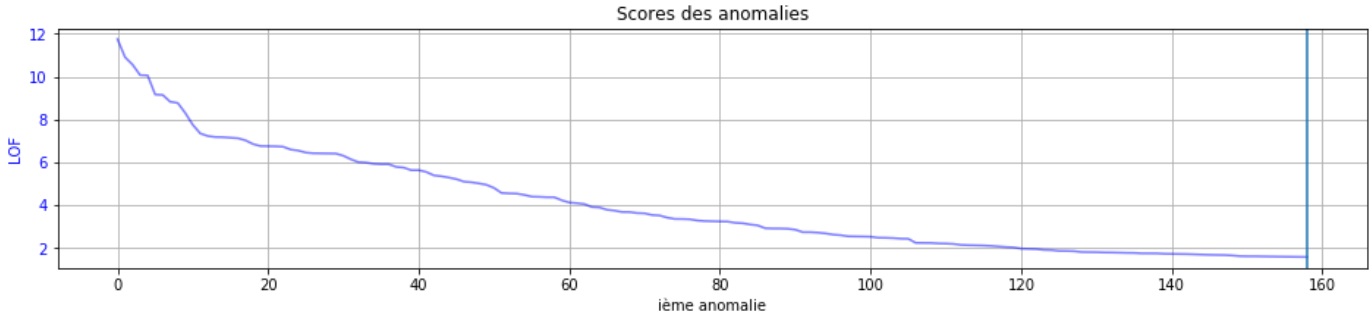

Fig. 11 : Scores obtenus par le LOF - Phase 1

On peut ici observer ci-dessus les scores obtenus avec ce modèle dans la phase 1, en choisissant les 3 features EB, CD, et MSM.

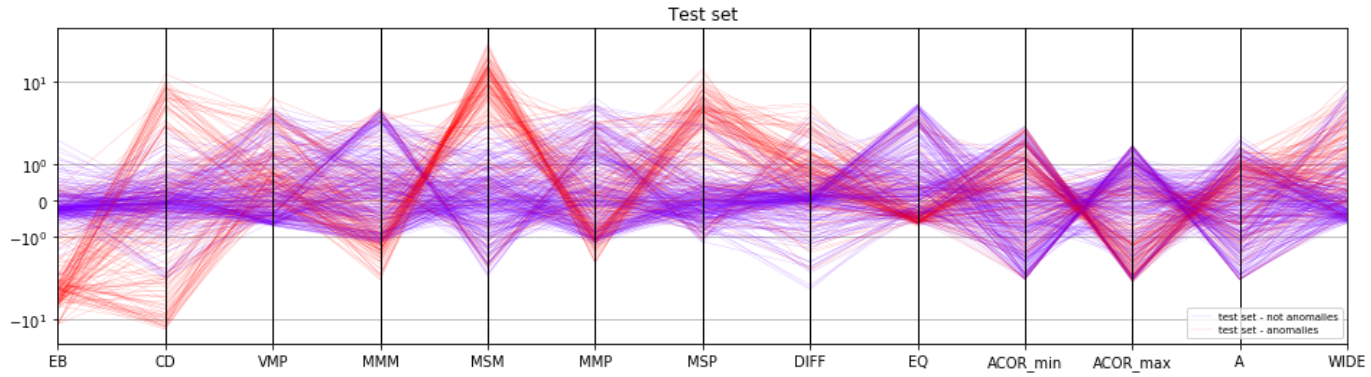

Fig. 12 : Anomalies détectées par le LOF - Phase 1

Le graphique ci-dessus, réalisé avec pandas.parallel_coordinates, donne une représentation des features des signaux par classe telles qu'elles ont été détectées par le LOF. Chaque signal est représenté par un ensemble de points reliés entre eux (un point par axe vertical correspondant à la valeur de la feature pour le signal donné). Les lignes associées aux signaux détectés comme anomalies par le LOF sont affichées en rouge, tandis que celles associées aux signaux détectés comme normaux sont violettes.

Le meilleur score obtenu lors de la phase 1 avec cette combinaison modèle / features est :

model	features	F1-score	Precision	Recall
LOF	EB, CD, MSM	**0,93761**	**0,93919**	**0,93603**

4.1.3 Isolation Forest

Cet algorithme est basé sur l'idée qu'une anomalie est plus simple à isoler de l'ensemble du jeu de données qu'une observation normale. Il fonctionne de la manière suivante. Pour chaque point du jeu de données :

- une feature est sélectionnée aléatoirement ;
- une valeur comprise entre la valeur minimale et la valeur maximale de cette feature est ensuite sélectionnée aléatoirement ;
- on partitionne alors les données selon cette valeur.

On réitère ces étapes jusqu'à ce que le point soit isolé du reste du jeu de données. On compte le nombre de partitions qu'il a fallu réaliser pour obtenir cette séparation. Du fait du caractère aléatoire des sélections de features et des valeurs, on recommence plusieurs fois l'intégralité du processus afin d'obtenir une moyenne du nombre de partitions nécessaires pour isoler le point d'intérêt. Les anomalies doivent en théorie être les points pour lesquels cette moyenne est la plus faible.

L'Isolation Forest n'est pas très sensible aux features corrélées, et ne fonctionne pas avec des calculs de distances. C'est pourquoi nous avons utilisé un plus grand nombre de features pour ce modèle (EB, CD, VMP, MMM, MSM, MMP, MSP, DIFF, EQ, ACOR_min, ACOR_max, A, WIDE).

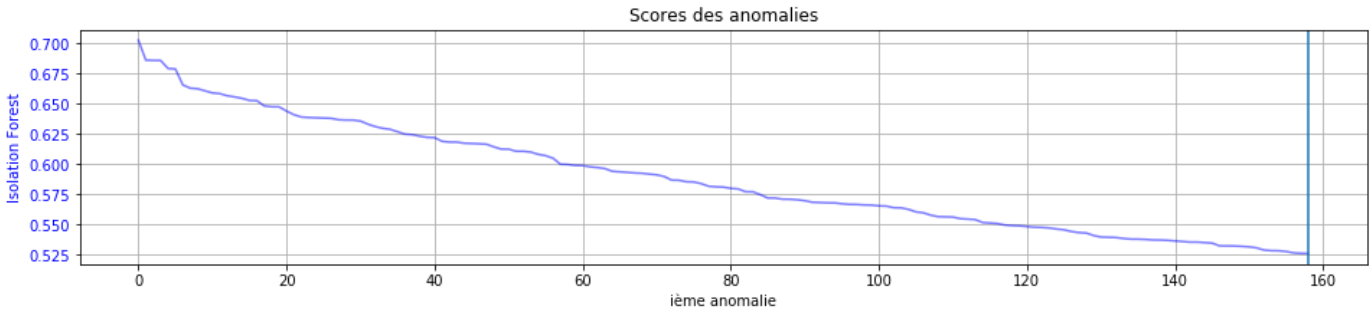

Fig. 13 : Scores obtenus par l'Isolation Forest - Phase 1

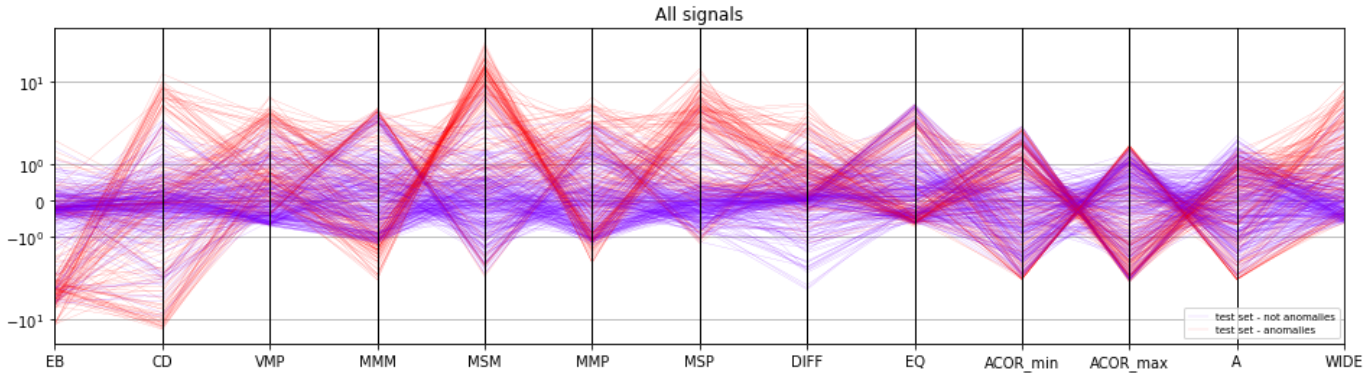

Fig. 14 : Anomalies détectées par l'Isolation Forest - Phase 1

Le meilleur score obtenu lors de la phase 1 avec cette combinaison modèle / features est :

model	features	F1-score	Precision	Recall
IF	EB, CD, VMP (+ 10 features)	**0,94416**	**0,94898**	**0,93939**

4.1.4 Gaussian Mixture Model

Le modèle de mélange gaussien (GMM) est un algorithme qui suppose que les observations ont été générées par le mélange de plusieurs distributions gaussiennes avec des paramètres inconnus. On suppose alors que toutes les observations d'une même distribution vont former un cluster dont les paramètres de distribution sont inconnus.

On commence par choisir un nombre k de distributions gaussiennes. Ensuite, pour chaque k, on cherche les paramètres de cette distribution (moyenne, écart-type). Pour chaque nouvelle observation, on regarde à quelle distance elle se trouve de chaque distribution puis on calcule la probabilité de cette observation d'appartenir à chaque cluster.

Si cette probabilité est faible pour tous les clusters, alors il est probable que l'observation n'appartienne à aucun des clusters et on la classe comme anomalie.

Pour ce modèle, nous avons choisi d'utiliser seulement 2 features afin de faciliter la visualisation. Après plusieurs essais, nous avons choisi les features EB et MSM, qui semblaient isoler le mieux visuellement un maximum de points du jeu de validation (voir figure ci-dessous), et qui ont effectivement donné les meilleurs résultats pour ce modèle. Le nombre de gaussiennes à utiliser

était jugé en regardant le graphique, souvent une seule ou deux, et on choisissait toujours une matrice de covariance par composante.

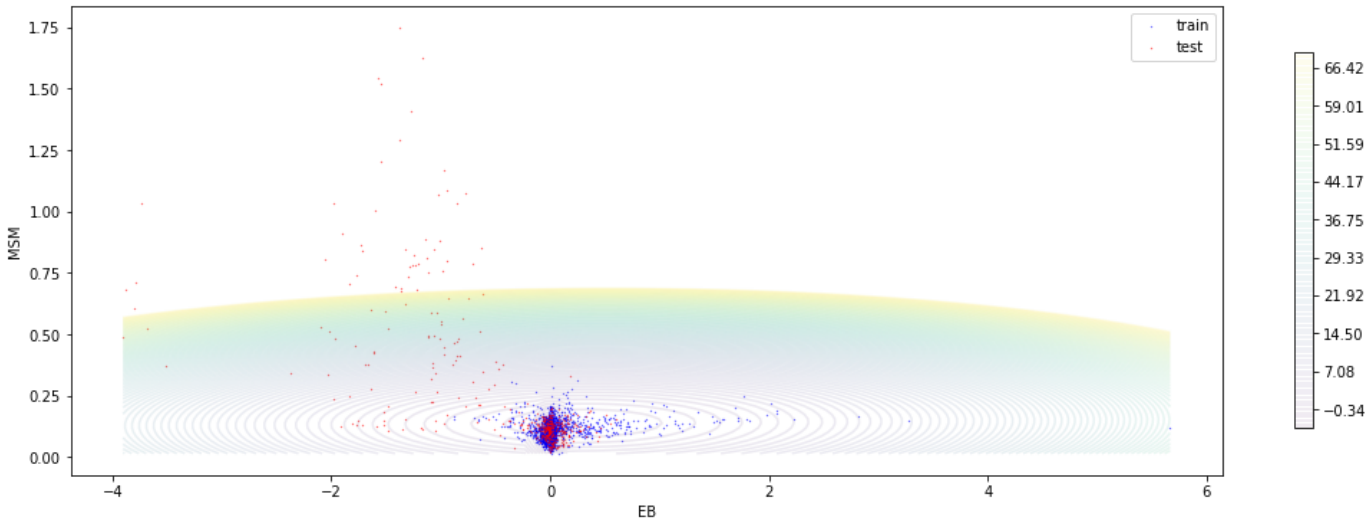

Fig. 15 : Représentation graphique des GMM

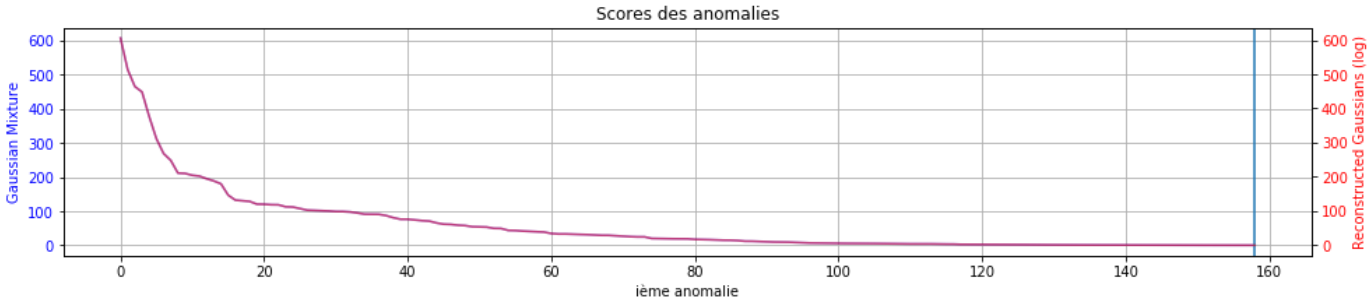

Fig. 16 : Scores obtenus par le GMM - Phase 1

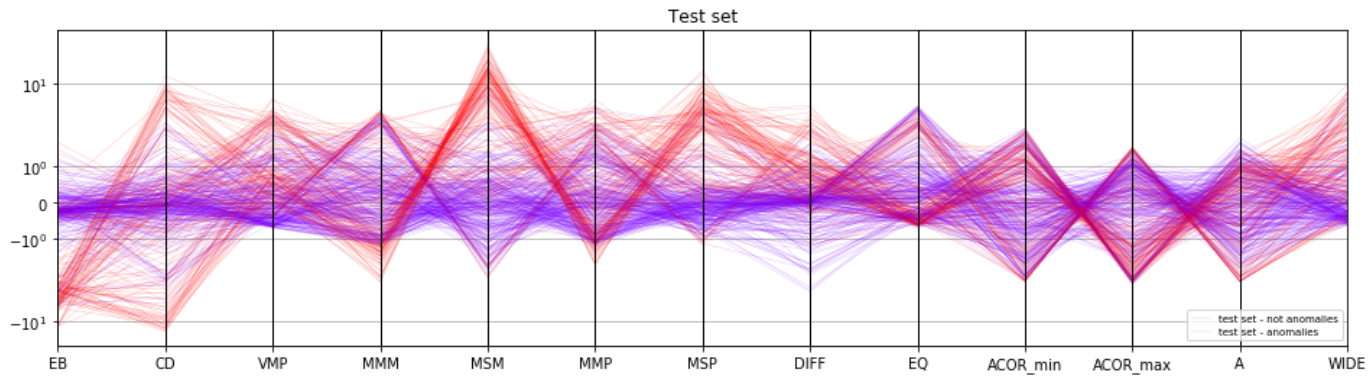

Fig. 17 : Anomalies détectées par le GMM - Phase 1

Le meilleur score obtenu lors de la phase 1 avec cette combinaison modèle / features est :

model	features	F1-score	Precision	Recall
GMM	EB, MSM	0,92256	0,92256	0,92256

4.1.5 Kernel Density Estimator

Le but du Kernel Density Estimator (KDE) est de généraliser la méthode d'estimation par histogramme, en y ajoutant notamment une propriété de continuité.

A cet effet, on crée pour chaque observation une gaussienne centrée sur cette observation. Toutes ces gaussiennes ont le même écart-type. L'estimateur à noyau de la densité (Kernel Density Estimator) est la fonction qui, à chaque X, associe la moyenne des gaussiennes en ce point. Plus il y a d'observations à proximité de X, plus la densité est élevée.

De plus, le KDE utilise un paramètre, appelé bandwidth, qui permet d'obtenir des courbes plus ou moins lissées. Plus ce paramètre est faible, plus la courbe est lissée. Ainsi, lorsqu'on cherche à détecter des anomalies, celles-ci sont supposées se trouver dans des zones de faible densité.

De la même façon, et pour les mêmes raisons que pour le modèle précédent, nous avons choisi d'utiliser seulement 2 features. Avec cette méthode, nous avons finalement, comme précédemment, choisi les features EB et MSM, qui semblaient isoler le mieux visuellement un maximum de points du jeu de validation (voir figure ci-dessous). Le choix du bandwidth était fait après quelques tests en regardant la représentation graphique ou en effectuant des tests.

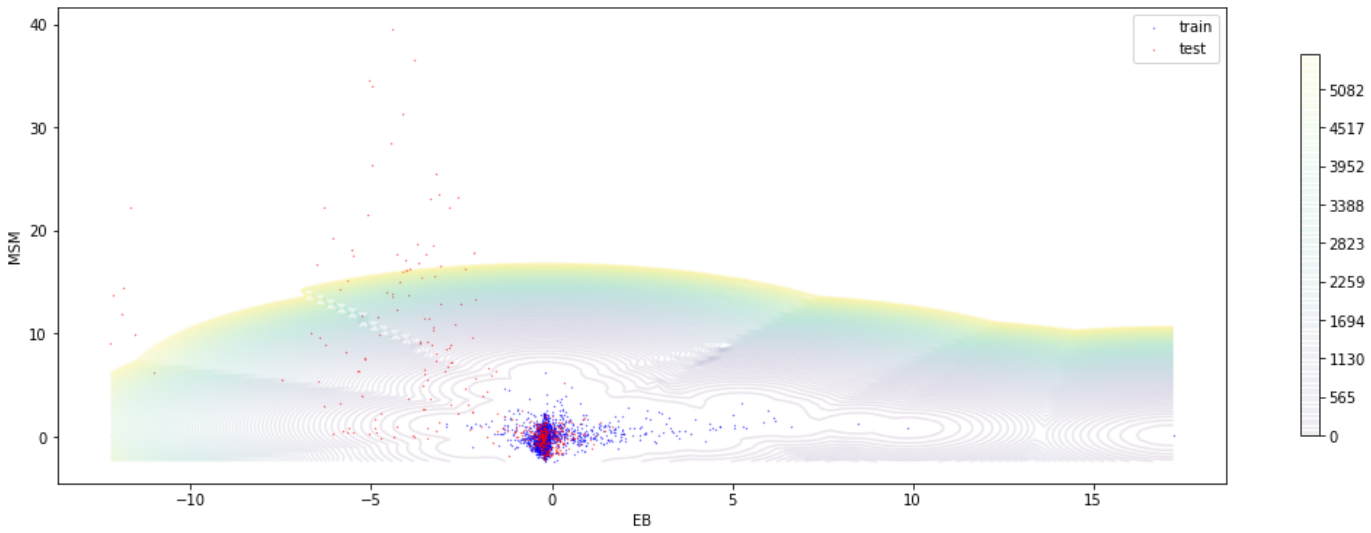

Fig. 18 : Représentation graphique du KDE

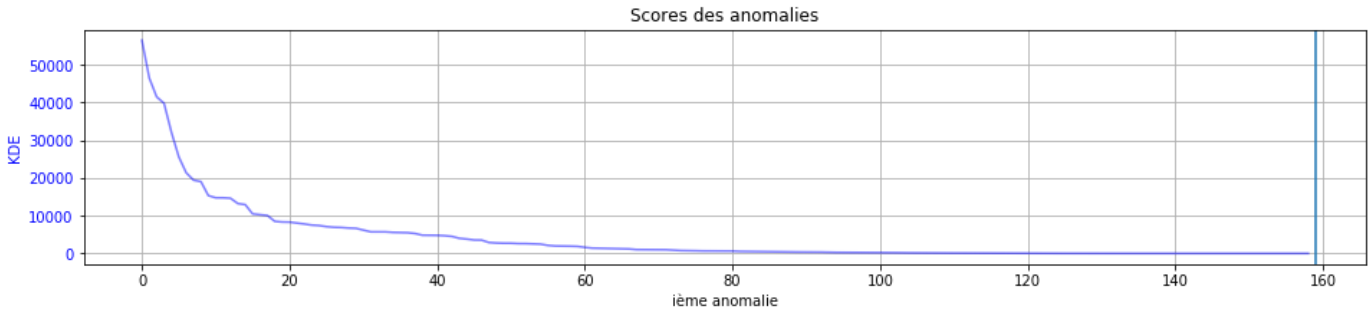

Fig. 19 : Scores obtenus par le KDE - Phase 1

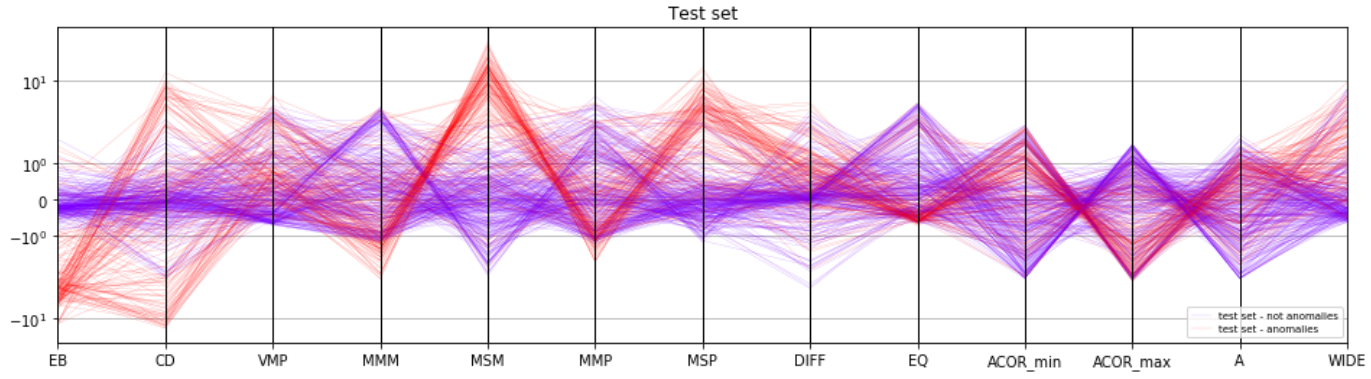

Fig. 20 : Anomalies détectées par le KDE - Phase 1

Le meilleur score obtenu lors de la phase 1 avec cette combinaison modèle / features est :

model	features	F1-score	Precision	Recall
KDE	EB, MSM	**0,92256**	**0,92256**	**0,92256**

4.2 Méthode de détermination du seuil

Les méthodes de détection d'anomalies que nous avons appliquées nous ont donné pour chaque signal un score. Il nous fallait ensuite choisir un seuil pour séparer les anomalies des signaux normaux. Les signaux ayant un score supérieur à ce seuil étaient classés comme anomalies et les signaux avec un score inférieur étaient classés comme sans anomalies. Pour déterminer ce seuil, nous avons testé plusieurs méthodes qui permettent de le fixer plus ou moins automatiquement, nous en exposons ici une qui mériterait d'être encore approfondie.

L'approche est la suivante : dans un premier temps, nous avons utilisé seulement une partie du jeu d'entraînement pour l'apprentissage, et évalué les scores des signaux restants, ainsi que les scores du jeu de test. En répétant de nombreuses fois cette étape et en changeant les données d'entraînement, nous avons pu obtenir pour chaque signal un score moyen.

Dans un second temps, nous avons calculé pour chaque seuil le nombre d'anomalies dans le jeu d'entraînement et le nombre d'anomalies dans le jeu de test. Ce faisant, nous avons pu tracer la courbe du pourcentage d'anomalies dans le jeu de test en fonction du pourcentage d'anomalies dans le jeu d'entraînement. Ce dernier étant considéré sans anomalies, et le jeu de test avec anomalies, on s'attendait à ce que cette courbe démarre à la verticale (0 anomalie dans le train, pour n dans le test), pour ralentir par la suite au fur et à mesure.

Dans un troisième temps, plusieurs choses ont été testées. D'abord en s'intéressant à cette courbe et à sa dérivée, qui était donc censée être une pente décroissante, nous avons essayé d'attraper le point d'inflexion où, lorsqu'on augmente le seuil, le pourcentage d'anomalies dans le jeu d'entraînement augmente bien trop vite par rapport à celui du test.

Nous avons également regardé le rapport du pourcentage des anomalies détectées dans le test divisé par celui dans l'entraînement pour chaque valeur de seuil, et ainsi fixé ce rapport qui était un indice de souplesse "s" (figure suivante, graphique en bas à droite, "s" = Value). Ce rapport

permettait de répondre à la question : "Quel est le seuil minimum qui nous donne au moins "s" fois plus de pourcentage d'anomalies dans le test que dans le train ?". Une façon pour trouver une valeur adéquate pour ce rapport était de la fixer au départ à une valeur élevée puis de la diminuer graduellement. En général, une valeur comprise entre 5 et 15 donnait de bons résultats. Cet ajustement était dépendant du modèle et du jeu de test, mais on pouvait souvent reporter cette valeur dans les autres calculs de seuils des autres modèles, en contrôlant quand même l'impact que ce choix avait sur le choix du seuil.

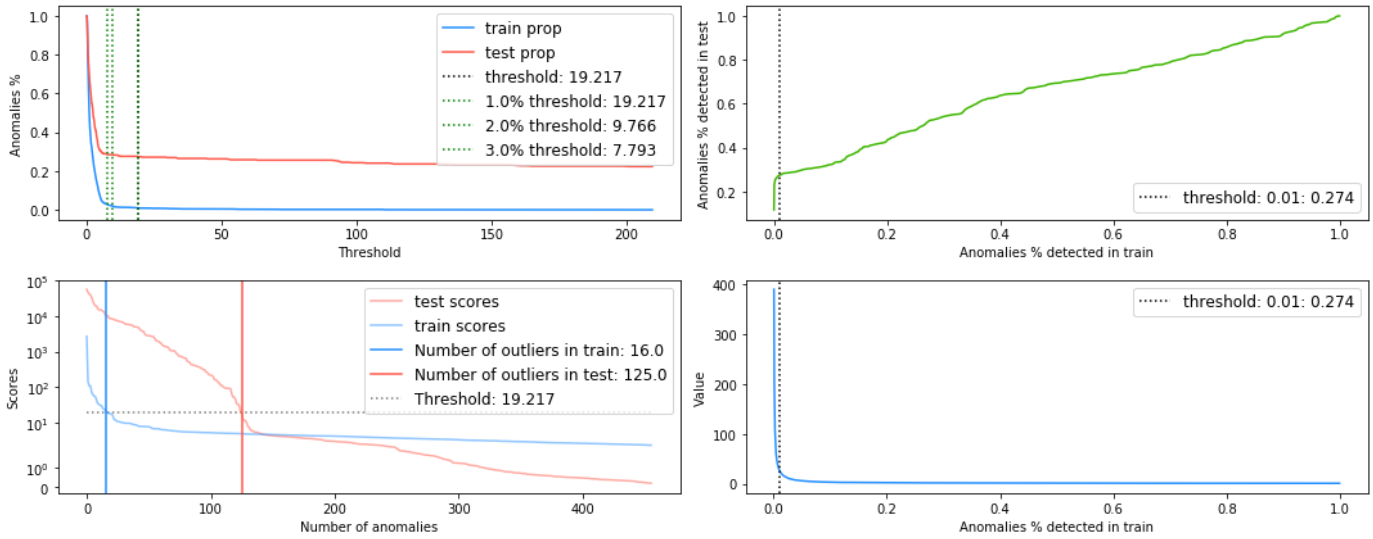

Fig. 21 : Visualisation de la méthode de détermination du seuil

Cette méthode n'est pas encore aboutie et mériterait d'être poursuivie, cependant elle nous a permis de trouver de très bons seuils très rapidement, grâce aux différentes visualisations.

Dans la première partie, nous pouvions facilement vérifier que notre approche fonctionnait puisque nous savions qu'il y avait 50% de signaux avec anomalies dans le jeu de test. Notre méthode de détermination de seuil devait donc sélectionner un seuil qui séparait le jeu de test pour qu'on obtienne environ 50% d'anomalies.

4.3 Méthode de validation

En plus de la méthode de cross-validation décrite précédemment, utilisée individuellement pour chaque modèle, nous avons décidé d'utiliser des méthodes d'ensemble qui combinaient les prédictions des différents modèles créés auparavant. Pour chaque signal, nous avons regardé combien de modèles le prédisaient comme anomalie. Nous avons ensuite affiché un graphique nous permettant de voir pour chaque nombre de modèles utilisés, combien il y avait de signaux avec anomalies détectés. En fonction des différents écarts de palier, nous avons décidé du nombre minimum de modèles ayant prédit un signal comme étant une anomalie pour le prédire comme tel.

#model	#anom	anom %
1	312	52.53 %
2	269	45.29 %
3	263	44.28 %
4	238	40.07 %

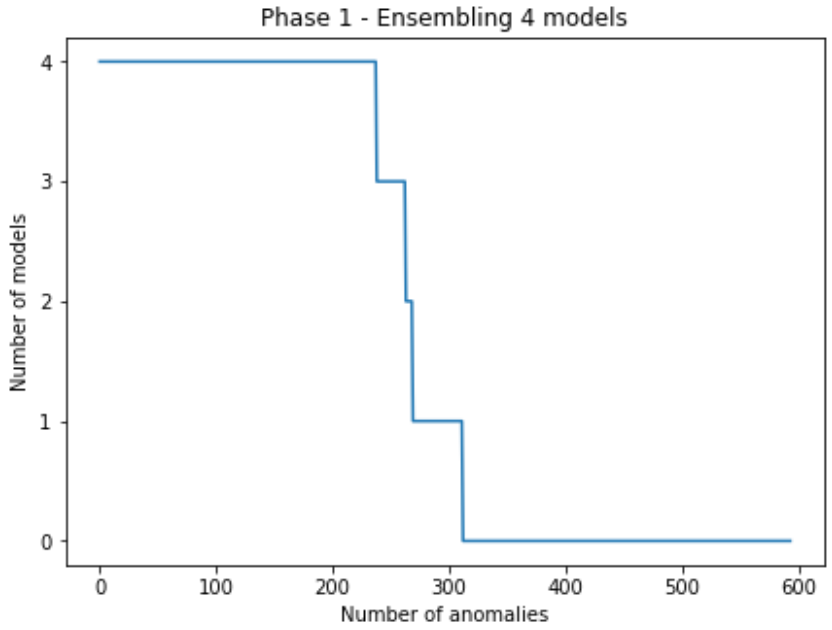

Fig. 22 : Nombre d'anomalies en fonction du nombre minimum de modèles nécessaires - Phase 1

Par exemple dans la phase 1, si un seul modèle parmi les quatre était nécessaire pour affirmer qu'un signal était une anomalie, nous obtenions 52.53% d'anomalies dans le jeu de test, contre 40.07% en les utilisant tous (voir figure précédente). En regardant les paliers, nous avons remarqué que choisir deux ou trois modèles parmi les quatre donnait des prédictions similaires. Nous avons finalement établi qu'il était nécessaire d'avoir au moins 50% des modèles, soit deux, pour classer un signal comme anomalie, ce qui permettait d'ajouter de la robustesse (par rapport à un seul

modèle nécessaire) sans trop être limitant. Ces méthodes d'ensemble nous ont ainsi permis de réduire la variance de nos prédictions et donc d'avoir une meilleure généralisation.

5 Résultats et phase finale

5.1 Résultats de la première phase

Pendant la première phase, nous avons utilisé un fichier Google Sheets pour récapituler les résultats obtenus par combinaison features / modèle, comme décrit en partie 1.2. Ci-dessous un extrait du tableau.

id	model	features	F1-score	Precision	Recall	anomalies_pred
1	LOF	EB, CD, MSM	0,93761	0,93919	0,93603	296
2	LOF	c3, longest_strike_above_mean	0,93155	0,92384	0,93939	302
3	IF	EB, CD, VMP (+ 10 features)	0,94416	0,94898	0,93939	294
4	KDE	EB, MSM	0,92256	0,92256	0,92256	297
5	GMM	EB, MSM	0,92256	0,92256	0,92256	297
6	GMM	max_MSEs, abs_energy	0,9215	0,93426	0,90909	289
7	Auto-encoder		0,90716	0,92256	0,89226	287
8	Distance to centroid	mean, std, range_above_under_	0,94416	0,94898	0,93939	293
9	Ensembling 1/3/5/8	-	0,97488	0,97	0,9798	300

Fig. 23 : Extrait du tableau récapitulatif de nos soumissions

Le meilleur score obtenu lors de cette phase est :

F1-score : 0.97488

Précision : 0.97 | Rappel : 0.9798

en assemblant les 4 modèles Local Outlier Factor, Isolation Forest, Gaussian Mixture et Distance to centroid. Le modèle KDE, semblable au GMM, ne nous a pas permis d'améliorer le score.

5.2 Stratégie pour la phase finale et résultats

Pour la phase finale, nous devions appliquer nos modèles sur un nouveau jeu de données : le jeu de test. Nous avons commencé par afficher les signaux pour voir si les potentiels signaux avec anomalies du jeu de test étaient semblables aux signaux avec anomalies du jeu de validation. Il s'est avéré qu'ils étaient assez différents. Nous avons pu confirmer cette intuition en utilisant nos outils

de visualisation de features. En particulier, nous avons constaté que notre feature Estimated Bias n'était plus assez discriminante, étant donné que les données n'étaient pas très décentrées.

Comme pour la première phase, nous avons ensuite décidé d'utiliser un ensemble de modèles pour déterminer si les signaux étaient anormaux ou pas. Nous avons repris les modèles de la phase 1, en utilisant nos visualisations et l'outil de seuil par cross-validation pour vérifier s'ils étaient toujours pertinents. Nous avons ainsi fait quelques ajustements au niveau des combinaisons features / modèles, en veillant à ce que les seuils choisis prédisent bien un très faible pourcentage d'anomalies dans le jeu d'entraînement (⩽5%), et permettent une séparation du jeu de test visible graphiquement (dans les modèles à 2 ou 3 features).
Les modèles que nous avons décidé d'assembler ainsi que les features sélectionnées pour chacun d'eux sont :

model	features
IF	EB, CD, VMP, MSM, ACOR_min, A, WIDE
IF	med, std, max_MSEs, c3, abs_energy
KDE	EQ, MSM
KDE	WIDE, CD, EB
LOF	EB, VMP, ACOR_max
LOF	c3, longest_strike_above_mean, abs_energy
LOF	med, std, max_MSEs
GMM	WIDE, EQ
Distance to centroid	mean, std, range_above_under_mean

Fig. 24 : Modèles et features - Phase 2

Contrairement à la première phase, le seuil correspondant au nombre de modèles ayant prédit un signal comme anormal a été plus difficile à déterminer. En effet, nous pouvons voir que les écarts de palier sont beaucoup plus petits que pour la première phase et aucun n'est réellement significatif par rapport aux autres. Il a donc fallu faire un choix arbitraire et nous avons décidé d'utiliser un nombre de modèles de 4 pour la première soumission et un nombre de modèles de 5 pour la seconde. Cela correspondait respectivement à 26.45% et 22.01% de signaux avec anomalie.

#model	#anom	anom %
1	1148	59.89 %
2	910	47.47 %
3	647	33.75 %
4	507	26.45 %
5	422	22.01 %
6	346	18.05 %
7	281	14.66 %
8	218	11.37 %
9	147	7.67 %

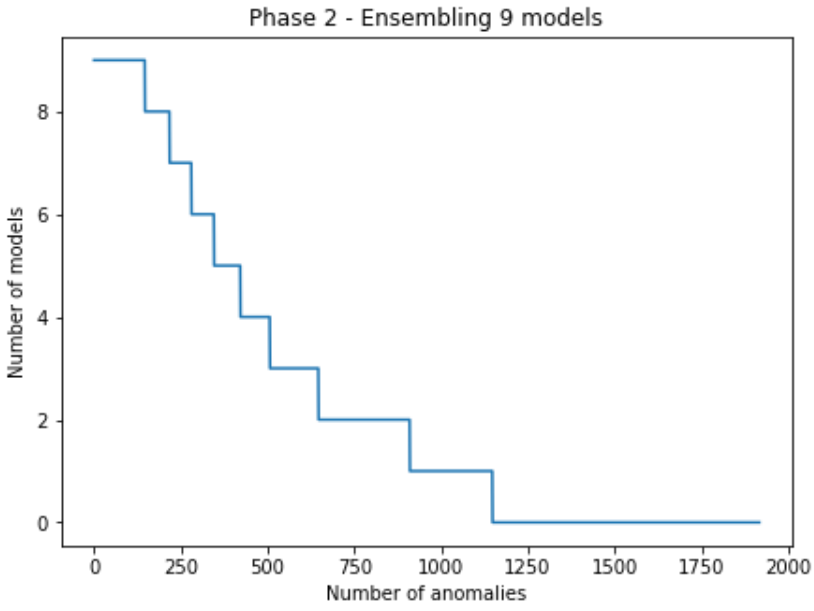

Fig. 25 : Nombre d'anomalies en fonction du nombre minimum de modèles nécessaires - Phase 2

Nous avons finalement obtenu les résultats suivants :

F1-score : 0.3664

Précision : 0.29985 | Rappel : 0.46190

Nous savions qu'il serait difficile d'obtenir un bon score lors de cette phase, mais nous avions tout de même confiance en nos méthodes de validation. Nous n'avons pas pu obtenir suffisamment d'informations lors de la cérémonie du 23 janvier pour nous permettre de mieux comprendre les données et les résultats. Nous avons cependant appris que le jeu de test contenait environ 20% d'anomalies, ce qui explique en partie pourquoi notre rappel est relativement élevé par rapport à notre précision. Nous aurions sans doute pu gagner en rappel et augmenter un peu notre F1-score en prédisant davantage d'anomalies (nous avons d'ailleurs remarqué que la plupart des équipes gagnantes ayant présenté leur travail avaient prédit autour de 50% d'anomalies), mais de toute manière notre modèle n'aurait pas été suffisamment robuste pour apporter une solution satisfaisante au problème.

Conclusion

Nous avons obtenu dans la phase finale un score bien moindre que dans la première phase. Les anomalies étaient très différentes dans chacune des deux phases et nous pensons que nos modèles ont eu du mal à reconnaître les nouveaux types d'anomalies.

Nous pensons qu'une meilleure connaissance du fonctionnement des hélicoptères et des capteurs nous aurait aidé à mieux appréhender le problème et à trouver davantage de features pertinentes. Cela pourrait être un point d'amélioration pour les prochaines éditions du défi : permettre une communication entre les étudiants et des experts métiers, ou donner davantage d'informations au début du projet : documentation, explications sur les jeux de données. Par exemple, nous savions que plusieurs séquences pouvaient provenir du même vol (et étaient toutes classées comme anomalies si l'une d'entre elles contenait une anomalie), sans savoir desquelles il s'agissait. Nous avons toutefois conscience que les autres équipes se sont heurtées à la même difficulté. En ce qui nous concerne, notre position dans le classement est restée stable d'une phase à l'autre (dans le top 30%).

Cependant, même si nous sommes déçus par notre résultat, nous avons beaucoup appris à l'occasion de ce défi. Ce fut pour nous l'opportunité de travailler en équipe sur un sujet s'apparentant à une recherche scientifique. De plus, nous avons découvert de nombreuses méthodes de détection d'anomalies utilisant aussi bien le machine learning, comme le Local Outlier Factor ou l'Isolation Forest, que le deep learning, avec l'auto-encodeur. Cela a également été pour nous l'opportunité de mettre en oeuvre des techniques de feature engineering, et d'apprendre à créer nous-même les features les mieux adaptées à un problème. Nous avons développé des compétences en détection d'anomalies, en traitement de séries temporelles et plus généralement en machine learning compétitif, et nous avons fait en sorte de rendre notre code facilement réutilisable si nous sommes amenés à traiter de nouveau des problèmes similaires.

Documents de référence

T. Hastie et al., *The Elements of Statistical Learning*, Second Edition (2008) :

https://web.stanford.edu/~hastie/Papers/ESLII.pdf

Documentation tsfresh - Feature extraction :

https://tsfresh.readthedocs.io/en/latest/api/tsfresh.feature_extraction.html

Site MLWave - Kaggle Ensembling Guide :

https://mlwave.com/kaggle-ensembling-guide/

Annexe

Architecture de l'auto-encodeur