Rapport de projet
Protection des données

Guillaume BEQUET
Karim DELLALI
Côme FRAPPE–VIALATOUX
Leonardo NASSABAIN

1 Introduction

Pour ce projet de protection de données, il nous a été demandé de mettre en place une chaîne d'analyse de données pour la cybersécurité, consistant à analyser les observations fournies puis d'implémenter plusieurs algorithmes de prédiction et de détection d'anomalies et de comparer les résultats. Nous avons choisi d'effectuer ces différentes tâches sur le jeu de données **SWaT**, décrivant les observations faites dans une centrale de traitement d'eau.

2 Jeu de données

2.1 Présentation

Le département ITrust de l'université de Singapour fournit chaque année depuis 2015 des données décrivant l'activité d'une centrale de traitement d'eau. La centrale est répartie en 6 unités de filtrage nommées de P1 à P6. Chaque seconde, les valeurs des différents capteurs et activateurs (77 au total) de l'usine sont enregistrées. Durant la capture des données, des attaques de différents types sont simulées.

Nous avons choisi de nous concentrer sur les données récupérées le 20 juillet 2019. Environ 4 heures d'observations sont à notre disposition.

Voici les 6 attaques qui ont été simulées (sur à peu près 30 minutes) :

1. Modification de la valeur de **FIT 401** de 0.8 à 0.5 afin de stopper la de-chlorination.
2. Modification de la valeur de **LIT 301** de 835 à 1024 dans le but de vider **T 301**.
3. Passage de **P 601** de OFF à ON pour augmenter l'eau dans le réservoir d'eau brute.
4. Passage de **MV 201** de CLOSE à OPEN et **P 101** de OFF à ON pour faire déborder **T 301**.
5. Passage de **MV 501** de OPEN à CLOSE pour drainer l'eau de RO.

6. Passage de **P 301** de ON à OFF afin d'arrêter l'unité 3 de l'usine.

2.2 Nettoyage

Les données n'étaient pas tout à fait conformes pour travailler après avoir importé le CSV. Pour y remédier nous sommes passés par différentes étapes :

— Labélisation du jeu de données.
— Suppression des colonnes numériques ou catégorielles pour lesquelles il n'y avait qu'une seule valeur possible.
— Vérifier s'il y avait des NaN dans les colonnes et les éliminer le cas échéant.

2.3 Standardisation

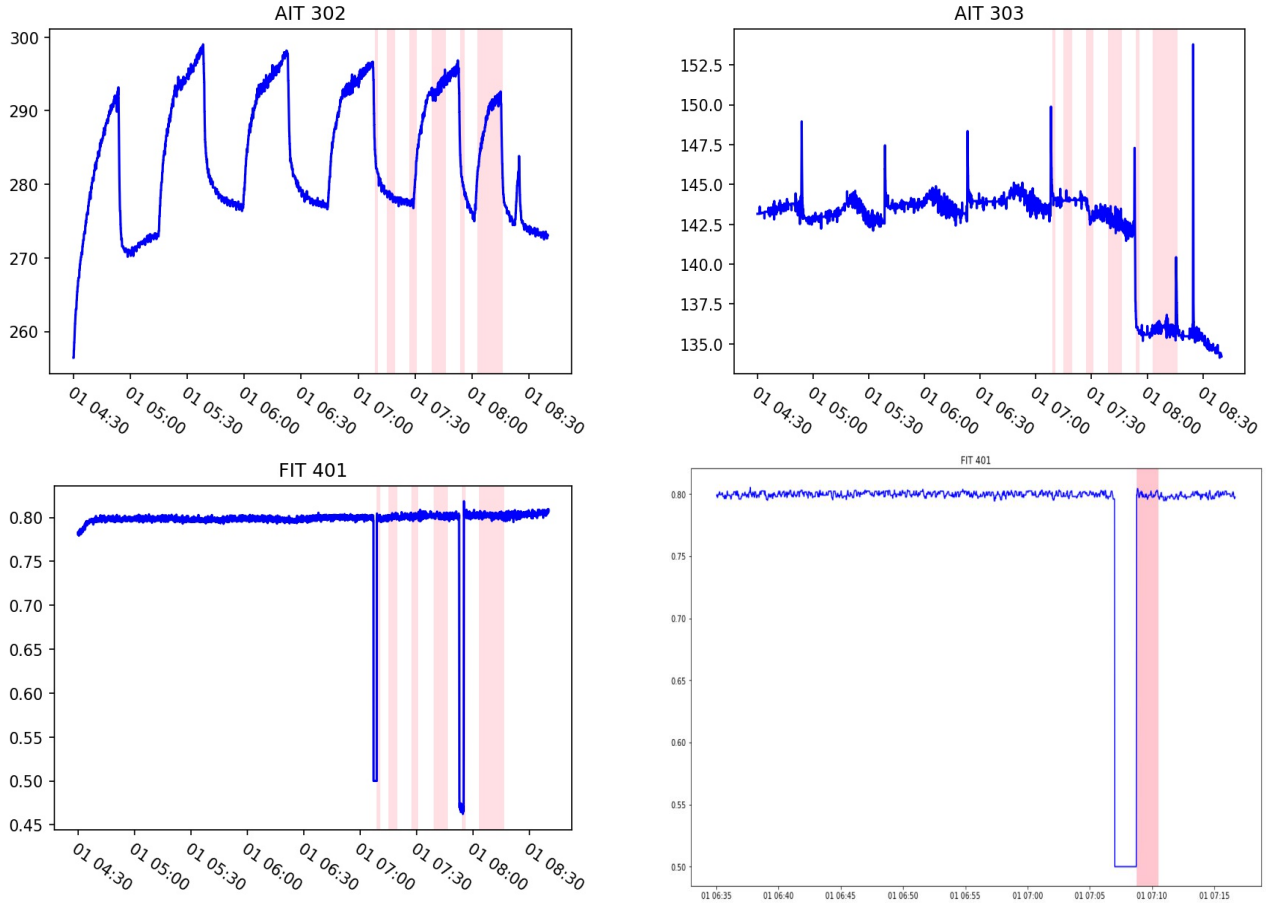


FIGURE 1 – Exemples de décalages observés lors de l'analyse des timezones d'attaque

La standardisation des données a été faite selon le traitement suivant :

— Vérification de la cohérence des labels entre la documentation et les données réelles
— Standardiser la colonne timeframe en ne gardant que l'heure, les minutes et les secondes
— Création de colonnes supplémentaires pour labéliser les différentes attaques
— Effectuer un encodage one-hot des colonnes catégorielles
— Remplacer les espaces dans les noms des colonnes par des '_'

A noter que nous avons rencontré des problèmes avec les labels des attaques. Dans le document officiel décrivant le jeu de données, les temps de début et de fin des attaques étaient fournis, mais après avoir

affiché les différents graphiques et examiné de plus près le jeu de données, nous nous sommes rendu compte qu'il y avait un décalage non négligeable entre les intervalles de temps fournis et les débuts et fins d'attaques réellement observées.

Pour remédier à cela, nous avons corrigé les labels en déterminant manuellement les bons débuts et fins de chaque attaque observée.

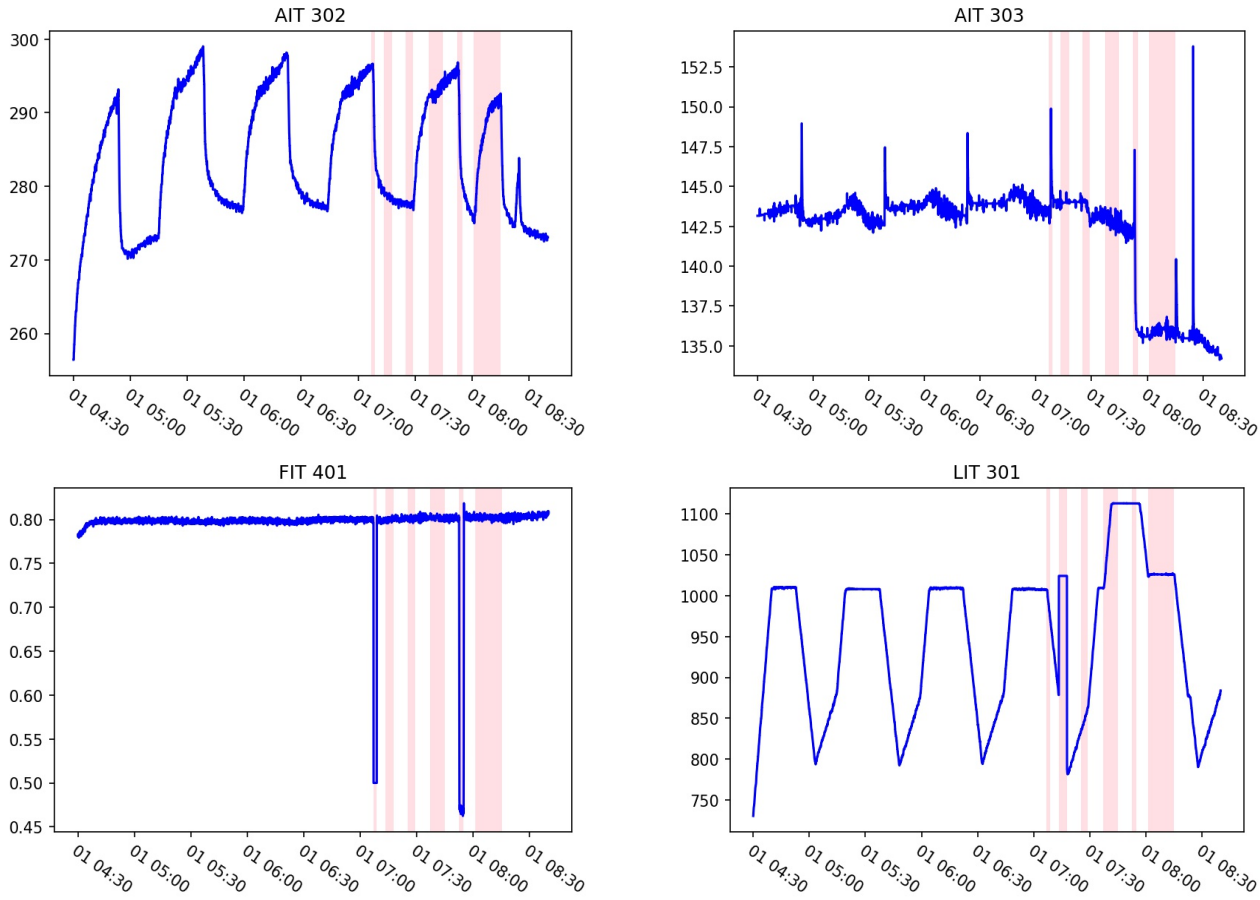
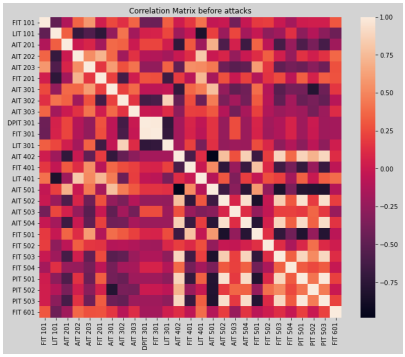


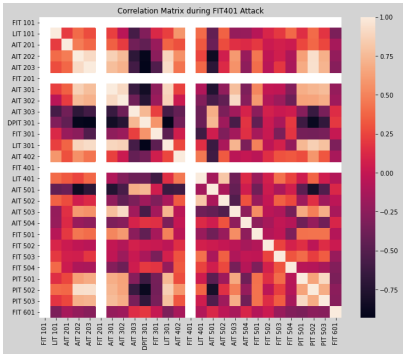
FIGURE 2 – Les différentes timezones d'attaque après correction

2.4 Matrices de corrélations

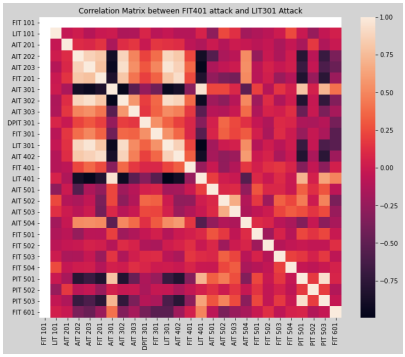
Nous avons créé plusieurs matrices de corrélation afin de comprendre comment chacune des attaques influence les différents paramètres. Nous avons créé une matrice de corrélation avec les données obtenues avant les 6 attaques, puis des matrices de corrélation pendant et entre chacune des attaques et enfin une matrice utilisant les observations faites après toutes les attaques ainsi qu'une sur l'entièreté des données. Nous voulions voir si les matrices de corrélation obtenues sur des observations entre les différentes attaques étaient similaires et s'il y a des corrélations récurrentes et communes aux différentes attaques.



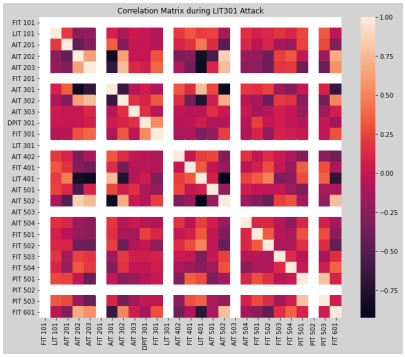
(a) Matrice de corrélation avant les attaques



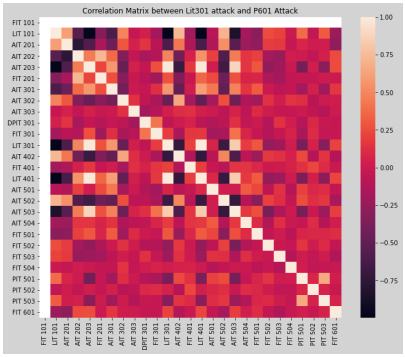
(b) Matrice de corrélation pendant attaque sur FIT401



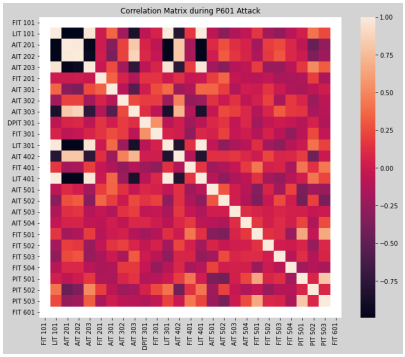
(c) Matrice de corrélation entre attaques FIT401 et LIT301



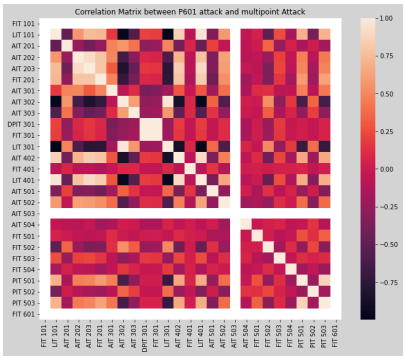
(d) Matrice de corrélation pendant attaque sur LIT301



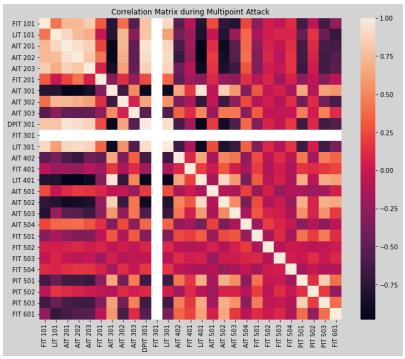
(e) Matrice de corrélation entre attaques LIT 301 et P601



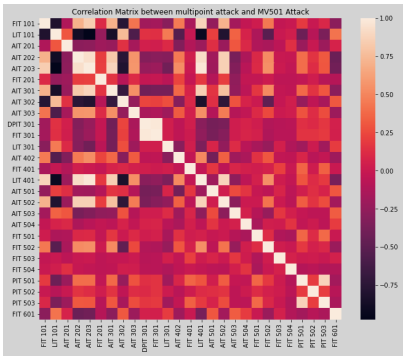
(f) Matrice de corrélation pendant attaque sur P601



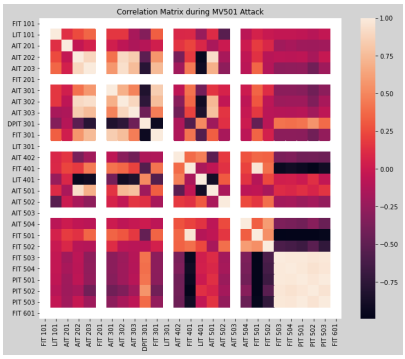
(g) Matrice de corrélation entre attaques P601 et Multipoint



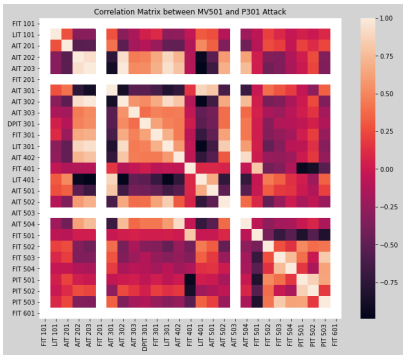
(h) Matrice de corrélation pendant attaque Multipoint



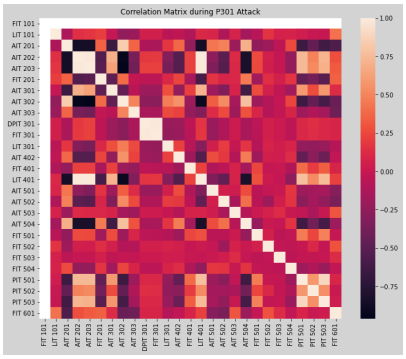
(i) Matrice de corrélation entre attaques Multipoint et MV501



(j) Matrice de corrélation pendant attaque sur MV501

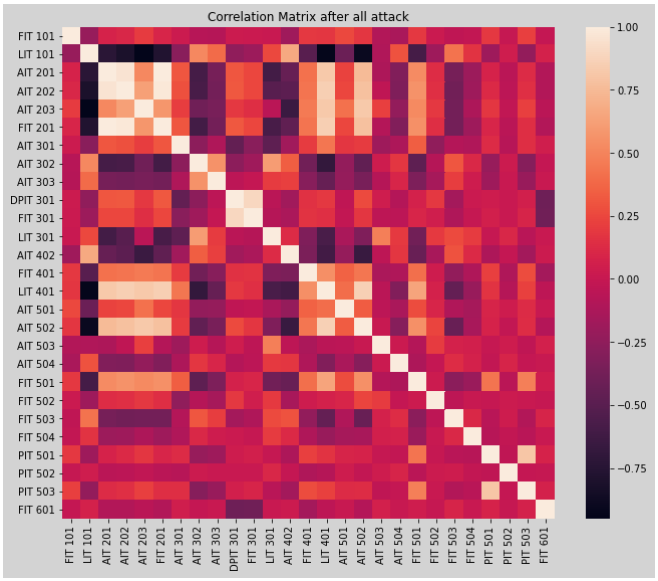


(k) Matrice de corrélation entre attaques MV501 et P301

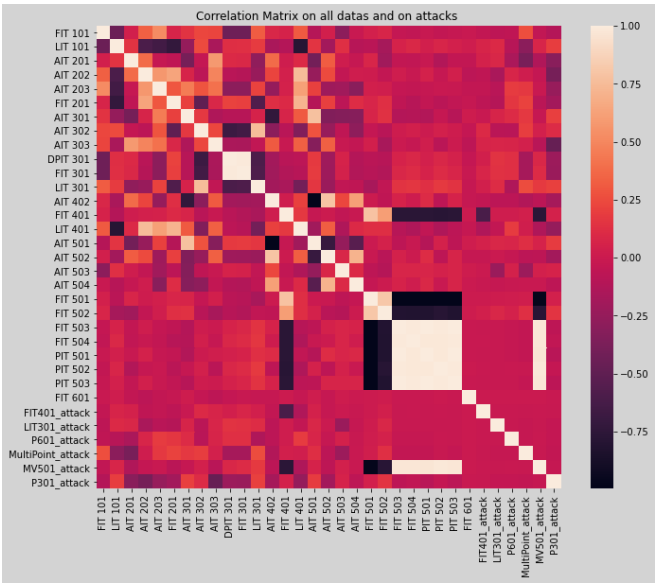


(l) Matrice de corrélation pendant attaque sur P301

FIGURE 3 – Matrices de corrélations



(a) Matrice de corrélation après les attaques



(b) Matrice de corrélation sur l'ensemble des données

FIGURE 4 – Matrices de corrélations

Cette vue d'ensemble des matrices de corrélation montre des corrélations très différentes selon le temps d'observation. Certaines ont des lignes blanches, ce qui signifie que l'attribut en question n'a pas bougé lors de la période de temps observée. Cela peut être dû à l'attaque, mais pas forcement, par exemple certaines attaques fixent les valeurs de certains attributs comme celle sur LIT301 3d. Ce qui est intéressant est d'observer lors des attaques, quels sont les attributs qui deviennent corrélés. Par exemple sur la figure (3j) qui s'étend sur la période de l'attaque sur MV501 on constate l'apparition de corrélations négatives. En analysant les attributs en question on remarque que compte tenu du fait que l'attaque induit le drainage d'un tank, ces corrélations traduisent les effets du drainage : les capteurs de flux sortant du tank sont négativement corrélés au niveau de celui-ci. Une approche émergeante de classification de l'état du système par matrices de corrélation sur fenêtre glissante pourrait donc possiblement être explorée.

2.5 ACP

Nous avons effectué une Analyse en Composantes Principales afin de réduire les 45 attributs en deux dimensions et afficher les données dans ce plan. Les 2 premières composantes principales restituent 92,94% de l'information contenue dans les données. L'ajout de la troisième composante principale compliquerait l'affichage sans apporter de nette amélioration quant à l'information contenue sur le graphique.

Nous avons supprimé l'attribut timestamp avant de créer l'ACP car il pourrait biaiser le calcul des composantes principales. Nous remarquons tout de même une continuité dans la répartition des points sur le plan. Nous pensons que ce phénomène est dû à la temporalité inhérente aux données et à la complémentarité des capteurs et activateurs présents dans l'usine.

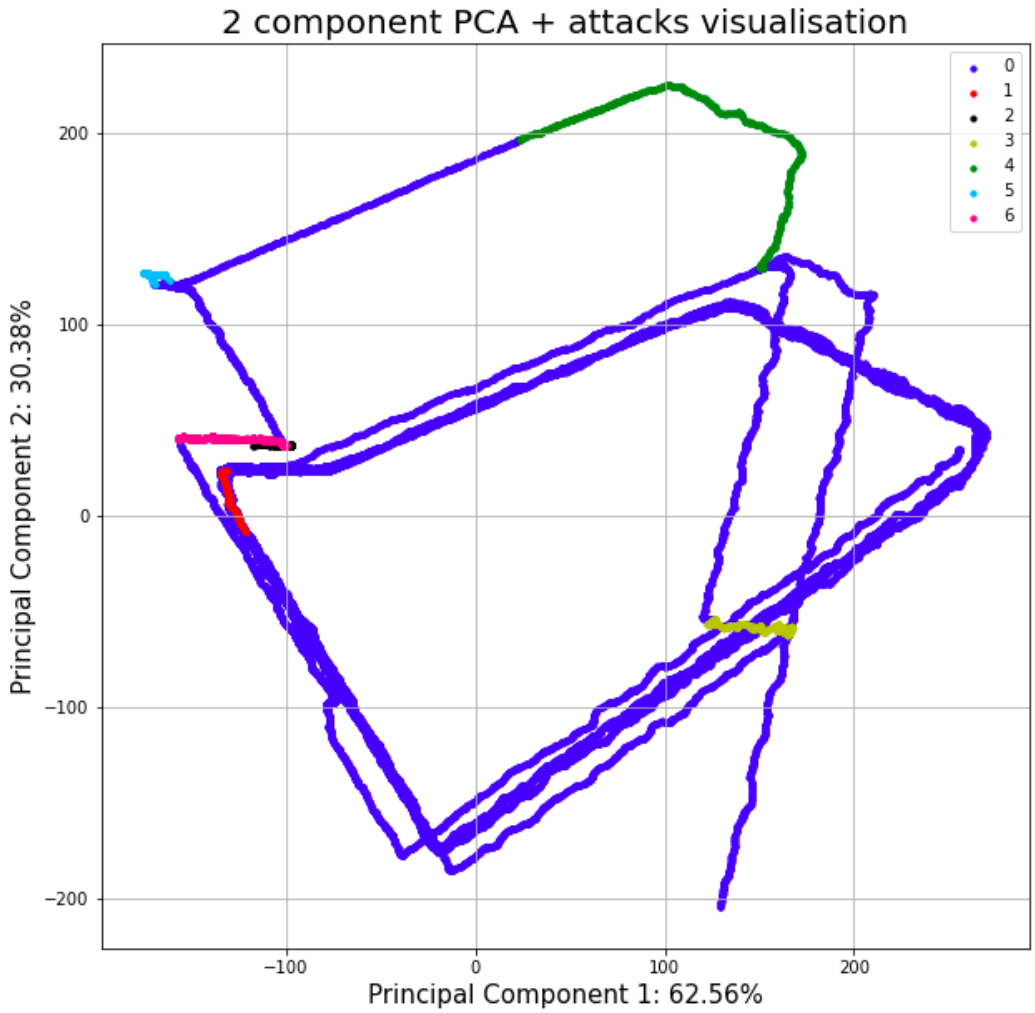


FIGURE 5 – ACP différenciant les attaques

Nous interprétons le rectangles formé de plusieurs lignes superposées comme un cycle de déroulement normal (sans attaques) de l'usine. En effet, lorsqu'un écart se fait il est souvent accompagné d'une attaque. Par exemple l'attaque 4 (la verte) modifie le trajet en "rectangle" initial. À la fin de l'attaque on voit que les données "repartent" dans la même direction que le rectangle.

Afin de vérifier cette hypothèse nous avons décidé de différencier les attaques selon différentes couleurs (cf Fig5). On remarque effectivement qu'elle se suivent bel et bien. Notamment les 3 - 4 - 5 - 6 (jaune - vert - cyan - rose) ce qui appuie l'idée de continuité. D'après l'ordre d'apparition de ces dernières, elles appuient un sens de lecture du temps sur le graphique de l'ACP comme suivant le sens trigonométrique (anti-horaire).

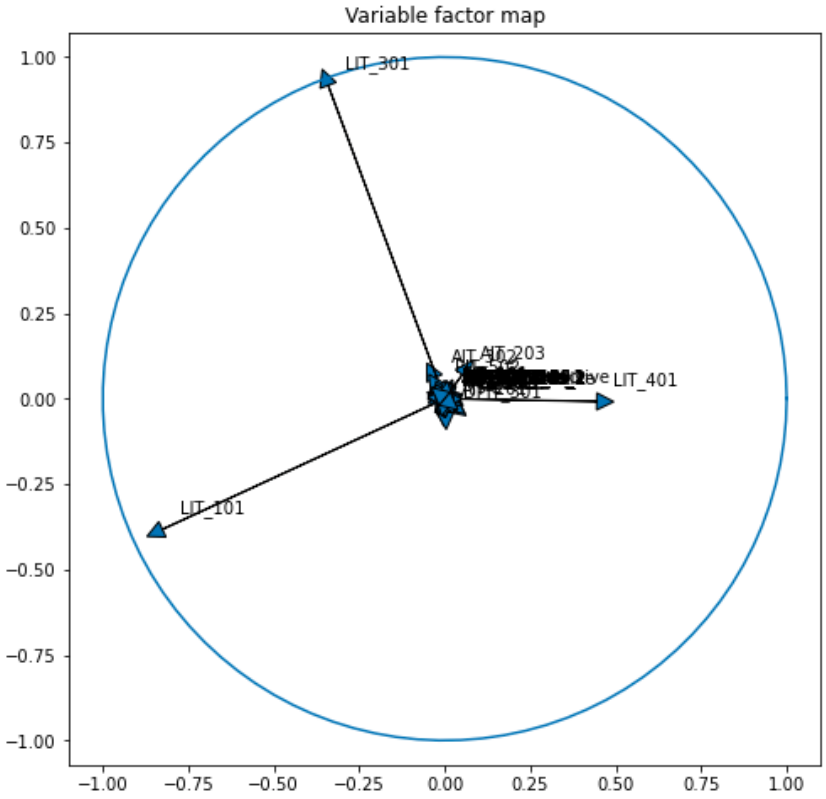


FIGURE 6 – Cercle de corrélation

On peut voir sur le cercle de corrélation de l'ACP que 3 attributs ont grandement influencé les composantes principales : **LIT 301** en haut à gauche, **LIT 101** en bas à gauche et **LIT 401** à droite. Nous ne comprenons pas vraiment pourquoi cela arrive à ces attributs en particulier et non à d'autres. Ce sont des remarques qu'il pourrait être pertinent de communiquer à un expert métier.

3 Algorithmes de machine learning : analyse et visualisation

Dans cette partie nous allons décrire les différentes approches de machine learning que nous avons utilisé pour analyser les données. Nous présentons aussi quelques mesures de performance et des visualisations des résultats obtenus. Pour tous les algorithmes nous avons préalablement supprimé la colonne timestamp qui aurait pu biaiser l'apprentissage.

3.1 Isolation Forest

Pour la mise en place de l'algorithme Isolation Forest, plutôt que de laisser chaque attaque en colonnes séparées dans le jeu de données, nous avons décidé de généraliser celles-ci en créant une colonne **isAttack**, dans laquelle on pourra les répertorier. Dès lors, quand une attaque aura lieu (que ça soit FIT401 ou LIT301 par exemple) on aura un label **isAttack** à 1, sinon 0.

Sur un total de 14 996 observations dans notre jeu de données, il y en a 2053 pour lesquelles **isAttack** vaut 1, ce qui nous donne un pourcentage de 13,69% d'outliers.

De ce fait, le classifieur utilisé sera le suivant :

```
clf = IsolationForest(max_samples = nb_samples, contamination=outlier_fraction,\
                      random_state=rng)
```

Avec comme valeur de contamination 0,1369, et random_state de 99, sur 2053 anomalies détectées nous avons 683 vrai positifs , soit un taux de détection de 33%.

3.2 Local Outlier Factor (LOF)

Pour l'algorithme de détection d'anomalies Local Outlier Factor, nous avons utilisé le classifieur suivant :

```
clf = LocalOutlierFactor(n_neighbors = 5000)
```

Pour le paramètre n_neighbors, qui décrit le nombre d'individus pris en compte pour évaluer la distance d'un individu par rapport à son voisinage, nous avons essayé plusieurs valeurs différentes et nous avons remarqué que pour n_neighbors=5000 le taux d'anomalies détecté, c'est-à-dire le ratio entre les vrais positifs et le nombre d'anomalies dans le jeu de données, était le plus élevé. Comme dit dans la partie précédente, le jeu de données comporte 14 996 observations dont 2053 sont faites lors d'une attaque et sont donc labélisées comme anomalies. Avec cet algorithme, 376 observations étaient classées comme anomalies dont 283 qui le sont vraiment. Donc nous avons un taux de vrais positifs qui est relativement élevé : 75%. Cependant, 283 anomalies détectées sur 2053 qui sont dans le jeu de données reste un taux de détection assez faible : 13.785%.

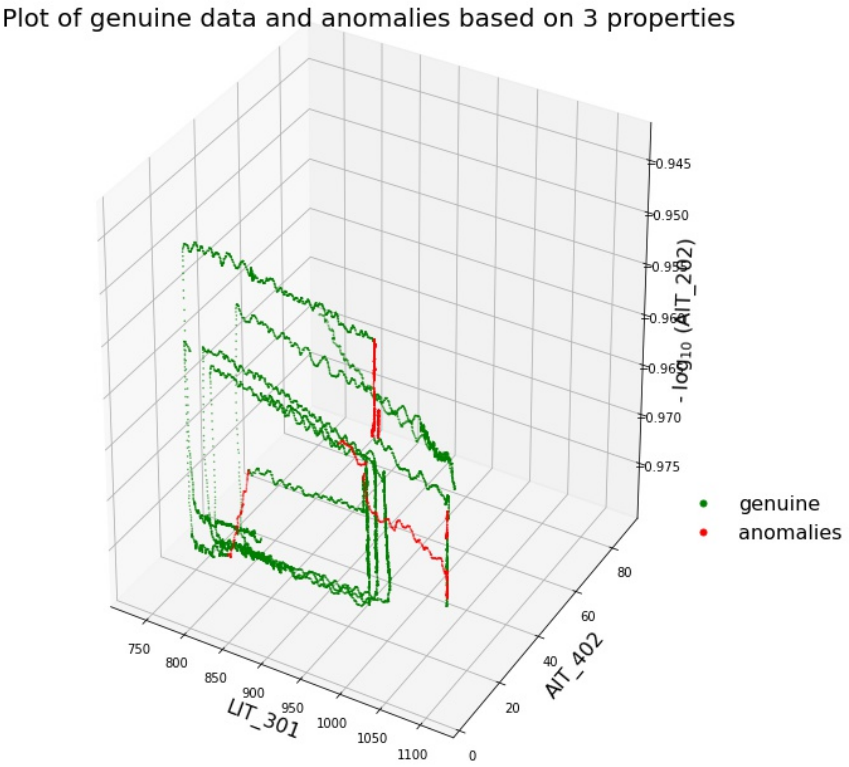


FIGURE 7 – Représentation 3D des données en fonction des valeurs de LIT 301, AIT 402 et AIT 202

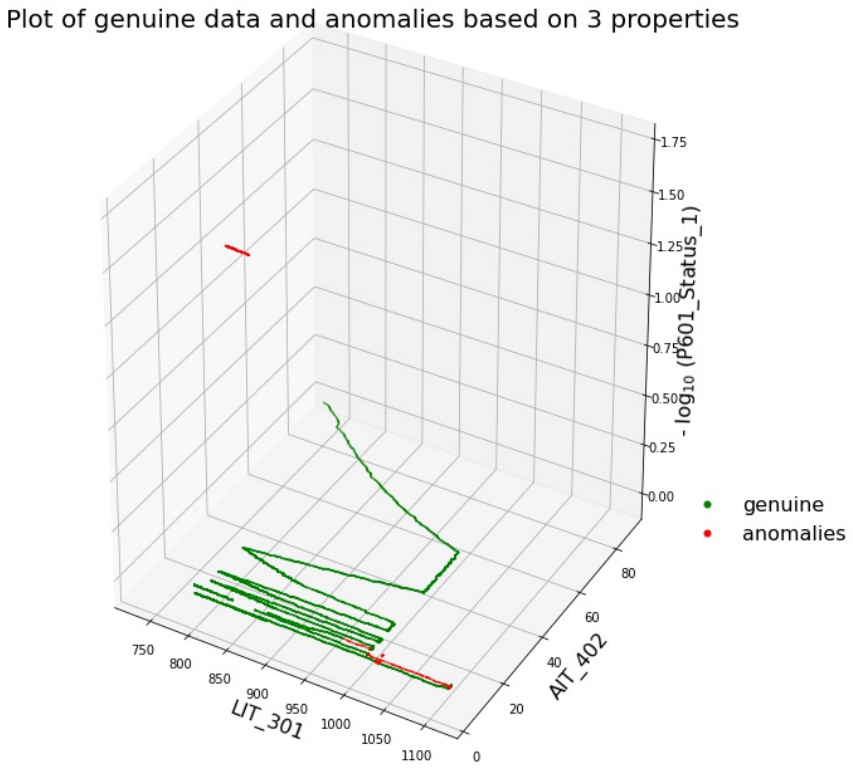


FIGURE 8 – Représentation 3D des données en fonction des valeurs de LIT 301, AIT 402 et P601_Status_1

Sur les figure 7 et 8 représentant le jeu de données en fonction de 3 attributs, nous pouvons voir les anomalies en rouge et les observations normales en vert. Suite aux analyse sur la couverture, le gain et le poids obtenus après l'entraînement XGBoost, nous avons vu que ces 4 attributs étaient les plus importants dans la classification par XGBoost. C'est pourquoi nous affichons les données selon ces axes.

Nous nous rendons compte que même s'il existe des anomalies qui sont éloignées du reste des données, la majorité reste relativement proche des données normales et peuvent donc facilement être mal classées. Ce qui explique pourquoi les taux de détections sont de 33% et de 13.785% pour Isolation Forest et LOF respectivement.

3.3 XGBoost

Contrairement à Isolation Forest et Local Outlier Factor, XGBoost est une méthode de prédiction d'anomalies et non pas de détection. Il s'agit donc d'un algorithme basé sur l'apprentissage supervisé de détection d'anomalies. Avant de commencer nous avons dû diviser les données en un jeu d'entraînement et un jeu de test. Pour faire cela, nous avons utilisé la fonction train_test_split de la librairie SciKit Learn et nous avons choisi d'utiliser 80% des données pour l'entraînement et les 20% restants pour la validation.
Ensuite, en sachant que les anomalies constituent uniquement 13,69% des observations, il fallait ajouter plus d'importance aux anomalies afin d'équilibrer le jeu de données et ne pas obtenir des résultats biaisés. Nous avons donc utilisé le paramètre scale_pos_weight de la fonction XGBClassifier en définissant que le poids des observations positives est le suivant :

```
weights = (Y['isAttack']=='0').sum() / (1.0*(Y['isAttack']=='1').sum()
clf = XGBClassifier(max_depth = 3, scale_pos_weight=weights, n_jobs = 4)
```

Le poids des anomalies est donc obtenu en divisant le nombre d'observations 'normales' par le nombre d'anomalies dans le jeu de données. De plus nous avons défini que la profondeur des arbres créés par XGBoost est limité à 3 et que 4 threads sont impliqués dans les calculs.
Les résultats obtenus sont très bons, sur 3000 éléments dans le jeu de validation il y avait 384 anomalies et en utilisant XGBoost, nous avons réussi à en classer correctement 382. Ce qui donne un taux de prédiction correctes de 99,48%.
Dans le but de comprendre la suite, nous affichons d'abord un exemple d'un arbre de décision créé par XGBoost.

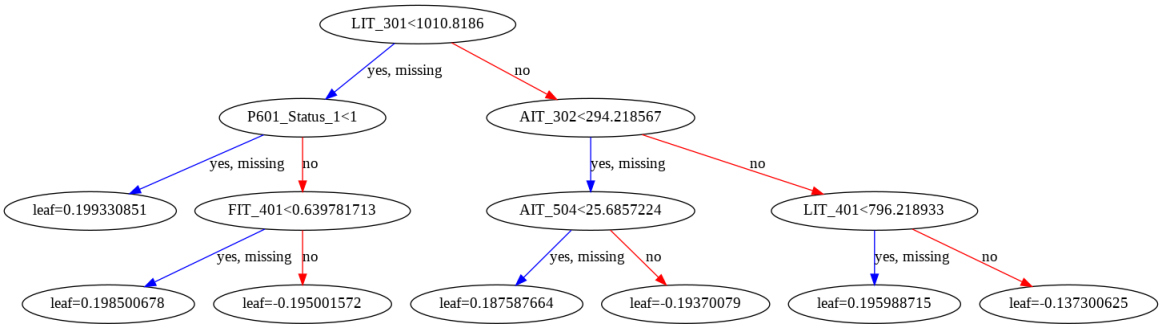


FIGURE 9 – Exemple d'arbre de décision crée par XGBoost

Nous pouvons voir dans ce graphique que dans chaque noeud de l'arbre un attribut est choisi et les observations sont discriminées selon leur valeur pour cet attribut. Pour effectuer une prédiction, XGBoost crée plusieurs arbres de ce type et effectue la décision finale en réunifiant les résultats de chacun de ces arbres. Pour exploiter et tirer des informations de ces arbres avec une méthode visuelle, nous avons décidé d'afficher l'importance des attributs selon 3 critères différents : le cover, le gain et le weight.

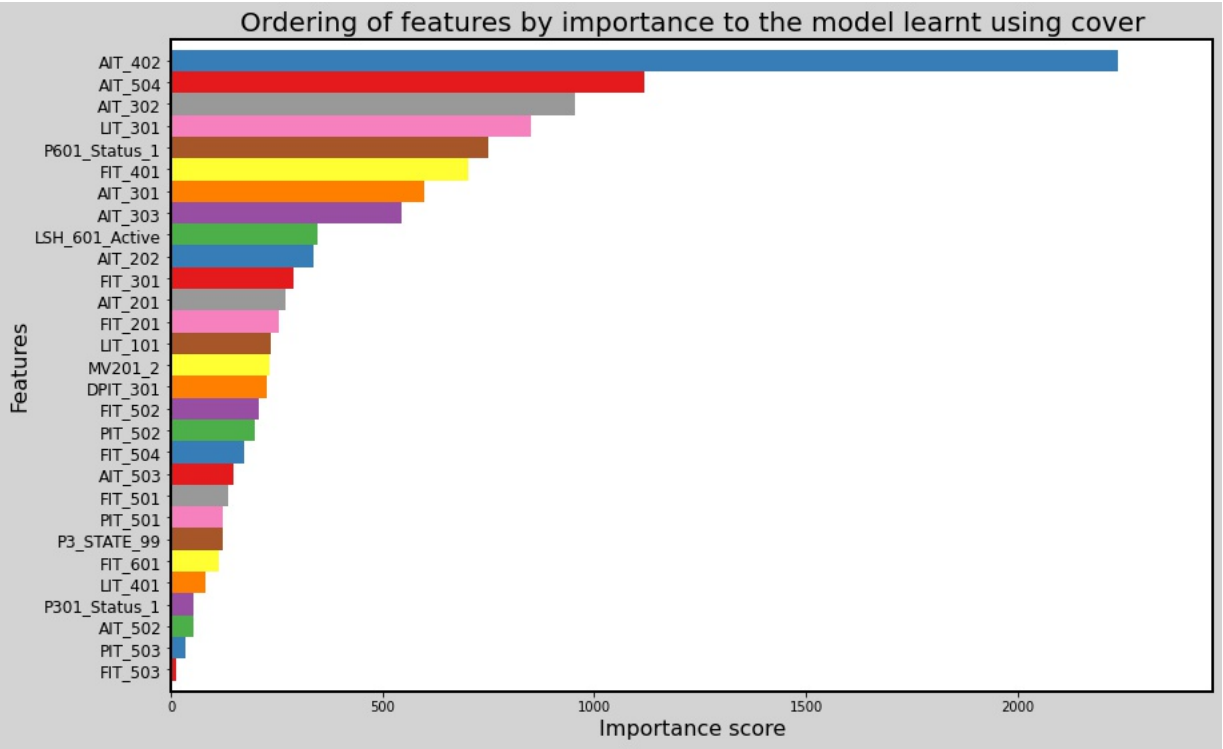


FIGURE 10 – Importance des attributs selon le cover

Le cover est mesuré en fonction du nombre de prédictions qui sont influencés par un attribut donné. Selon ce graphique, nous pouvons voir que c'est l'attribut AIT 402 qui influence le plus les prédictions. D'après la description du système de traitement d'eau, nous savons que AIT 402 correspond à l'analyseur de NaHSO3 et de NaOCl dans la partie P4 du système.

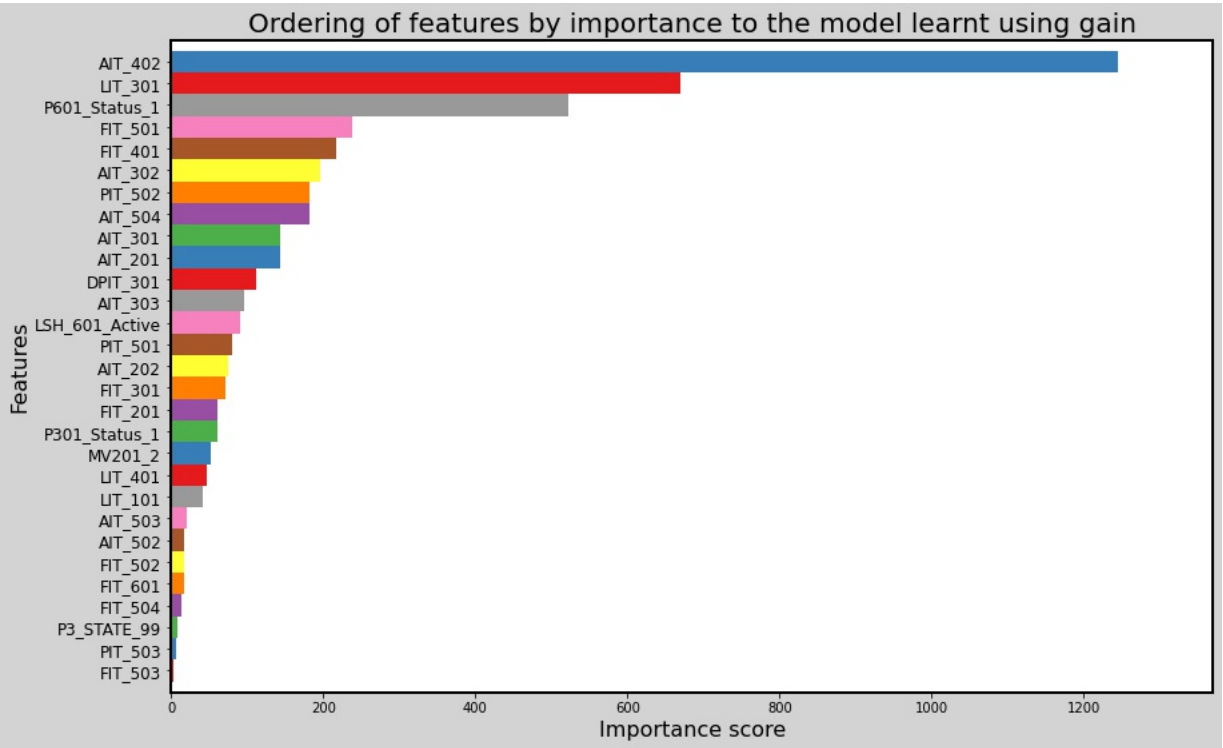


FIGURE 11 – Importance des attributs selon le gain

Le gain d'un attribut quantifie le pouvoir discriminant de cet attribut. C'est-à-dire l'importance et le poids de sa valeur dans la prédiction d'un label. D'après ce graphique nous pouvons donc voir que l'attribut le plus discriminant est AIT 402.

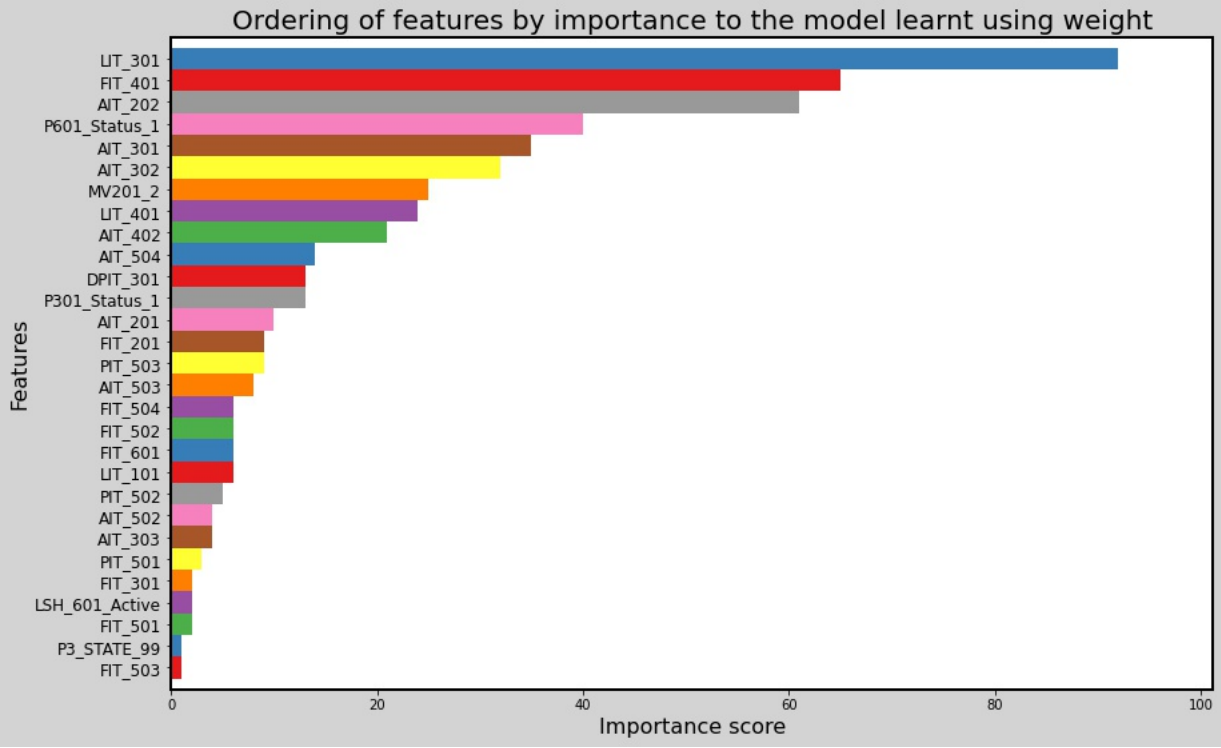


FIGURE 12 – Importance des attributs selon le weight

La valeur du weight d'un attribut est obtenue à partir du nombre de fois qu'un attribut est utilisé dans un split d'un des arbres utilisés par XGB. Donc selon ce graphique, l'attribut participant au plus grand nombre de splits, ou divisions dans le noeud d'un arbre est LIT 301. Cet attribut correspond au détecteur de niveau d'eau dans le UF feed tank de la partie P3 du système de traitement.

Par conséquent, il serait intéressant de surveiller de plus près et potentiellement avec des détecteurs plus fins la fluctuation des valeurs de AIT 402 et LIT 301 pour renforcer la sécurité du système et mieux détecter les attaques.
De plus, nous avons décidé d'afficher les courbes AUPRC et AUROC pour visualiser la qualité de prédiction de notre classifieur XGBoost.

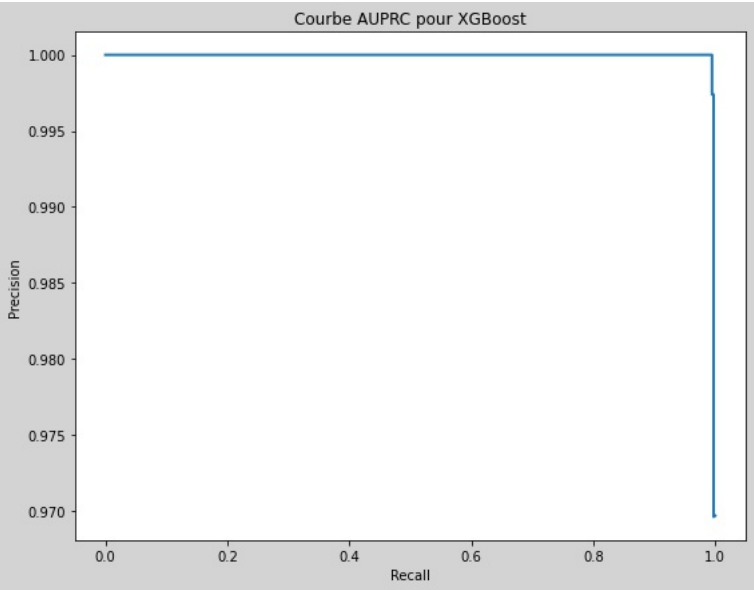


FIGURE 13 – La courbe PRC obtenue pour XGBoost

La courbe PRC visualise la capacité du classifieur à identifier les observations positives sans prédire trop de faux positifs. Donc notre modèle affiche une très bonne performance avec un AUPRC de 99.9%.

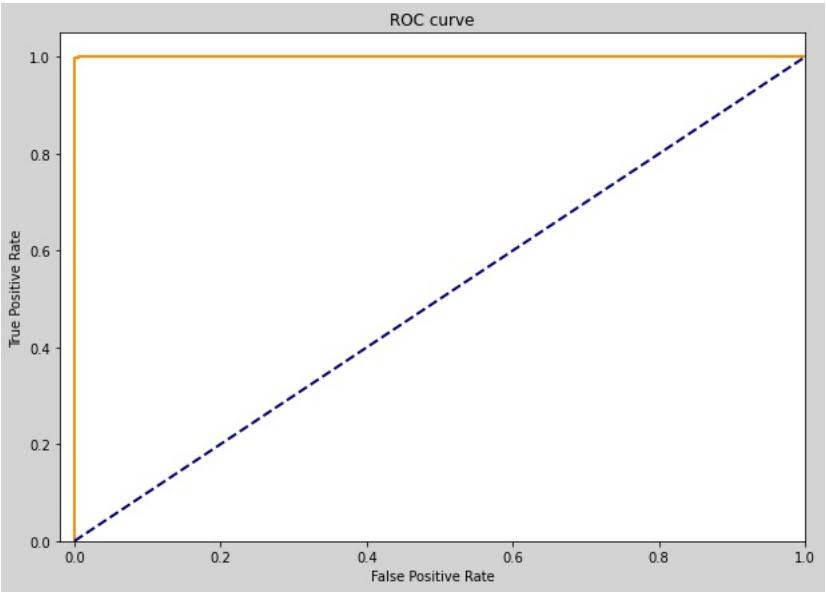


FIGURE 14 – La courbe ROC obtenue pour XGBoost

La courbe ROC décrit la probabilité qu'une anomalie choisie au hasard sera plus probablement classée comme anomalie qu'une observation normale choisie au hasard. Sur le graphique, nous voyons que la courbe ROC est bien au dessus de la valeur ROC d'une classification aléatoire (trait bleu pointillé) et donc nous pouvons conclure que notre classifieur est performant.

En conclusion de cette section, nous pouvons noter l'importance du capteur AIT 402, ce dernier se situant à la fois en haut du classement des attributs par gain et par cover. Cette importance se traduit par 2 aspects : il est capital pour la méthode de prédiction par XGBoost, et est donc particulièrement sensible aux attaques, car si ce dernier en est la cible, alors la méthode de détection elle-même s'en

retrouvera affectée. Les recommandations que nous pourrions donner seraient alors de le surveiller particulièrement pour s'assurer que ce dernier n'est pas cible d'attaque, et éventuellement d'ajouter un second capteur en redondance afin de prévenir les conséquences d'une mise hors service.

3.4 Random Forest

L'algorithme Random Forest est également une méthode de prédiction d'anomalies basée sur un apprentissage supervisé. Le principe de cet algorithme est de mettre en jeu un grand nombre d'arbres aléatoires, et de se servir de l'entropie pour discriminer, savoir quels sont les meilleurs.

Celui-ci construit un grand nombre d'arbre et fonctionne par vote majoritaire pour trouver la classe de chaque instance.

Tout comme pour XGBoost, nous avons également divisé notre jeu de données à l'aide du train_test_split en utilisant également 80% des données pour l'entraînement et le reste pour la validation.

```
clf = RandomForestClassifier(max_depth=i, random_state=0)
```

Pour le paramètre max_depth, nous avons choisi une profondeur maximale allant de 2 à 5, et pouvons ainsi mesurer l'évolution des performances de notre modèle selon les différentes valeurs de profondeur.

Le modèle est parvenu à détecter sur 3000 éléments du jeu de données (les mêmes utilisés avec XGBoost), un total de 376 anomalies sur les 384 répertoriées, soit une prédiction correcte de 97,91%. Ce résultat est très satisfaisant, bien qu'il demeure légèrement inférieur à XGBoost

A l'aide des graphiques présents sur la prochaine page, on peut observer l'évolution de la courbe ROC du modèle RandomForest, qui s'améliore nettement en fonction de la profondeur maximale que l'on passe au modèle. Nous avons obtenu comme résultat pour une profondeur de 2 un résultat de 0.7031 et à la fin avec une profondeur de 5 nous sommes parvenus à obtenir une valeur de 0.9895.

Comme expliqué précedemment pour l'algorithme ci-dessus, la courbe ROC décrit la probabilité qu'une anomalie choisie au hasard sera bien classée en tant que telle. Nous n'avons pas souhaiter augmenter à plus de 5 la profondeur, étant donné qu'on arrivait à un point stagnant dès lors que l'on passait ce cap.

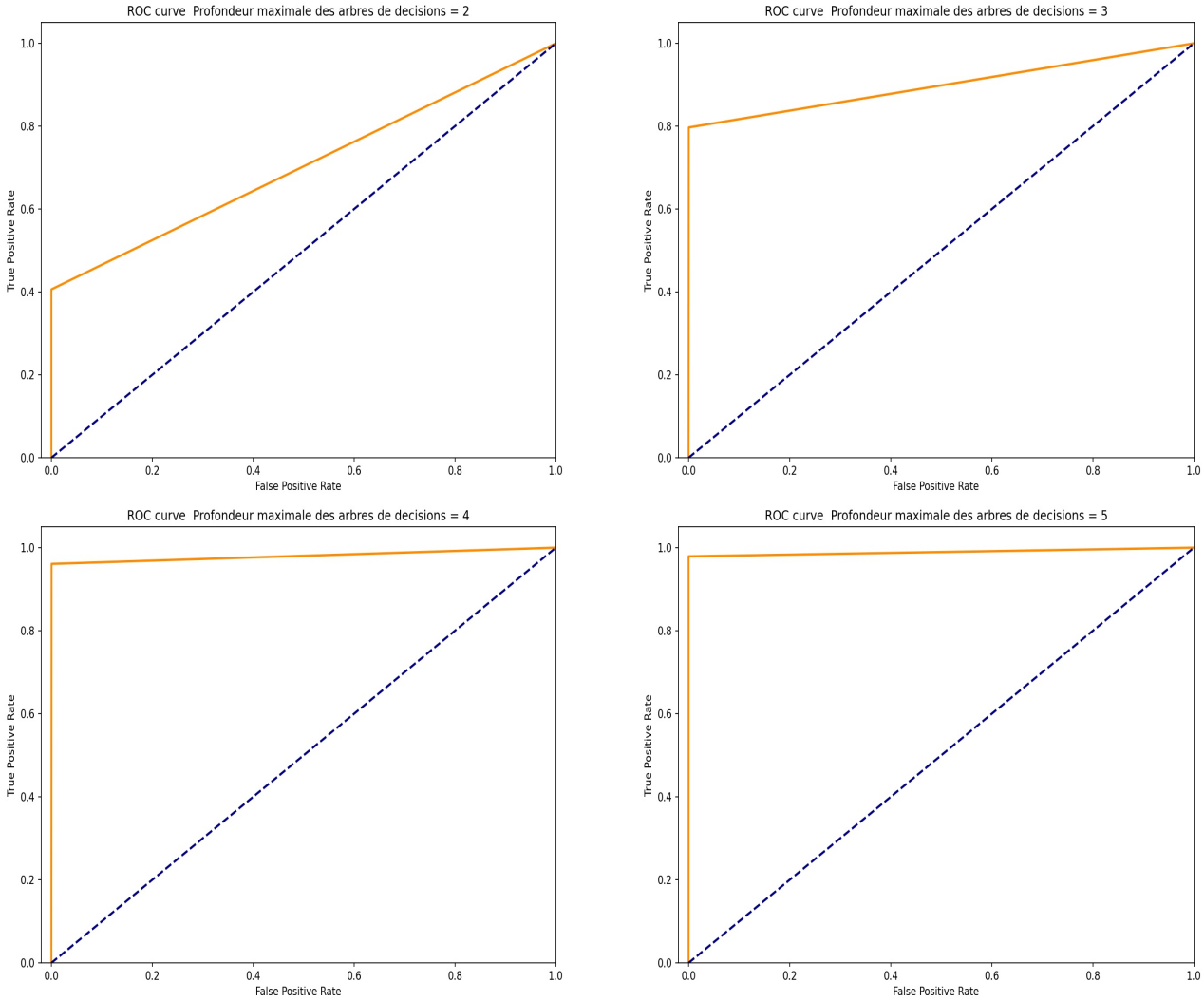


FIGURE 15 – Évolution de la courbe ROC en fonction de la profondeur maximale

3.5 KMeans

Afin de détecter précisément le type d'attaque nous avons eu l'idée de mettre en oeuvre un KMeans avec 7 clusters (les 6 attaques et les données normales). Comme les index que KMeans attribue aux clusters qu'il découvre ne sont pas forcément les mêmes que les attaques, il est difficile de vérifier que le clustering s'est bien déroulé. En affichant les clusters de KMeans sur le plan de l'ACP on remarque qu'ils ne correspondent pas du tout aux vraies attaques (cf Fig16).

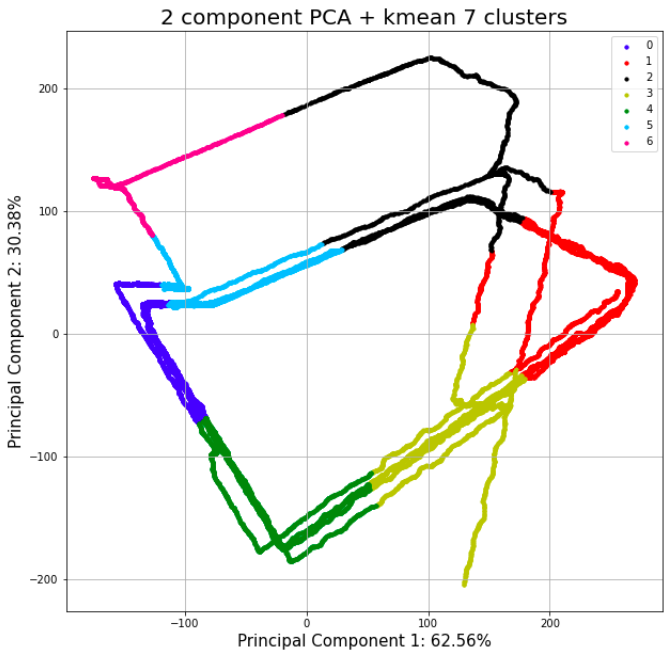


FIGURE 16 – Les 7 clusters de Kmeans

Nous avons aussi essayé de détecter simplement 2 clusters avec KMeans (attaques et non attaques). Les résultats ne sont pas non plus satisfaisants. (cf Fig17).

Par la suite nous avons également essayé d'appliquer cet algorithme sans grande conviction, simplement pour vérifier ce que nous pensions déjà : KMeans n'est pas adapté au données que nous traitons ici. En effet KMeans cherche des clusters mais pas forcement des données isolées.

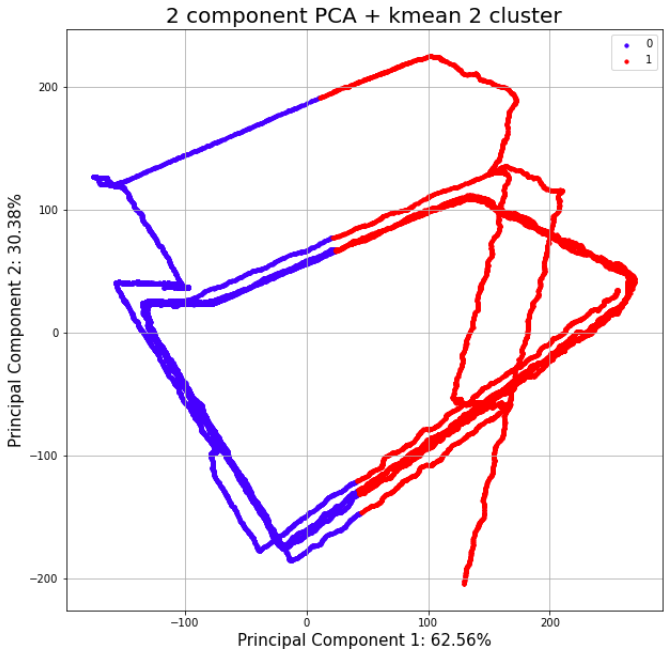


FIGURE 17 – Kmeans avec 2 clusters

3.6 LSTM

Pour tirer le plus parti de l'aspect temporel des données nous avons implémenté un réseau de neurones récurent, un LSTM (Long Short Term Memory) pour être exact (cf Fig18). Pour séparer les données d'entraînement et de test nous n'avons pas coupé le jeu de données en deux comme cela est souvent montré dans les exemples que nous avons pu trouver. Ce choix vient du fait que les 3 premières heures de capture des données ne contiennent pas d'attaques, tandis que la fin oui. Afin qu'il y ait autant d'attaques dans le jeu de test que dans celui d'entraînement nous prenons 8 données sur 10 pour l'entraînement et les 2 restantes pour la validation et on parcourt ainsi l'entièreté du jeu de données. À la suite de cette opération le jeu d'entraînement comportera 80% des données et le jeu de test 20%.

Nous utilisons une fenêtre de 30 secondes et un horizon de 5 secondes. Ainsi le RNN prédira s'il y aura une attaque dans les 5 secondes à venir en tenant compte des 30 dernières secondes.

Nous avons entraîné notre modèle sur 50 époques avec une possibilité d'early stopping dans le cas où on augmenterait ce nombre.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 30, 400)           444800
_________________________________________________________________
dense (Dense)                (None, 30, 20)            8020
_________________________________________________________________
bidirectional_1 (Bidirection (None, 300)               205200
_________________________________________________________________
dense_1 (Dense)              (None, 20)                6020
_________________________________________________________________
dense_2 (Dense)              (None, 20)                420
_________________________________________________________________
dropout (Dropout)            (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 105
=================================================================
Total params: 664,565
Trainable params: 664,565
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 18 – Architecture du LSTM

Malheureusement comme nous avons décidé de tester cette approche en fin de projet nous n'avons pas eu le temps de mesurer la performance de notre LSTM. Cependant au vue de la courbe d'apprentissage nous pouvons supposer que l'entraînement du réseau a été un succès (cf Fig19). En effet les erreurs dans le jeu de test et d'apprentissage suivent la même tendance et arrivent à un point de stabilité. La courbe de validation ne remonte pas ensuite ce qui aurait été un signe de sur-apprentissage. De plus les courbes ne semblent pas continuer à descendre ce qui aurait pu indiquer un sous apprentissage.

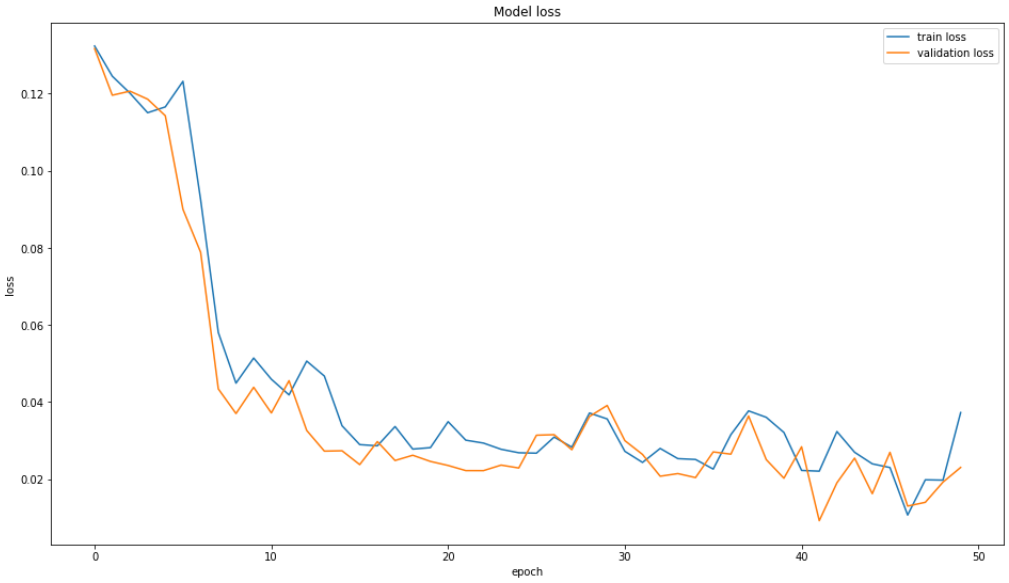


FIGURE 19 – Courbe d'apprentissage du LSTM

4 Comparaison d'algorithmes

Dans cette partie Benchmark, nous comparerons entres eux les algorithmes de prédiction et ceux de détection. En premier lieu il s'agira de la comparaison des algorithmes Isolation Forest et Local Outlier Factor, puis nous discuterons des algorithmes XGBoost et Random Forest. Nous regardons le taux de bonnes prédictions, les true-false/positive-negative ainsi que les scores d'AURPC et ROC AUC

Algorithme ML	Taux de prédiction	tn	fp	fn	tp	AUPRC	ROC AUC
XGBoost	0,99	2616	2	0	382	0,99	0,98
Random Forest	0,97	2616	7	0	376	0,98	0.98

TABLE 1 – Comparaisons entre les modèles de prédiction d'anomalies

Algorithme ML	Taux de détection	tn	fp	fn	tp	AUPRC	ROC AUC
Isolation Forest	0,33	11550	1393	1393	660	0,11	0,38
LOF	0,13	12850	93	1770	283	0,12	0,43

TABLE 2 – Comparaisons entre les modèles de détection d'anomalies

Comme expliqué précédemment, on observe de meilleurs résultats pour XGBoost lorsqu'on s'intéresse aux algorithmes de prédiction d'anomalies, bien que cela soit très léger, les deux algorithmes évalués ici produisent d'excellents résultats. En ce qui concerne les algorithmes de détection d'anomalies, aucun des deux étudiés n'est vraiment performant. En effet Isolation forest obtient plus de vrai positifs mais pour autant il a beaucoup plus de faux positifs que LOF, ce qu'il faut surtout observer est qu'aucun de ces deux algorithmes n'est réellement approprié aux données en question.

5 Conclusion

Pour conclure, après avoir essayé plusieurs algorithmes de prédiction et de détection d'anomalies et avoir comparé les résultats obtenus, nous nous rendons compte qu'il est beaucoup plus efficace de mettre en place des algorithmes avec une approche supervisée comme XGBoost et Random Forest. Comme nous pouvons le voir dans la partie 4 **Comparaison d'algorithmes** les taux de prédiction/détection sont nettement meilleurs pour les algorithmes XGBoost et Random Forest que pour Isolation Forest et LOF : 99% et 97% comparés à 33% et 13% respectivement.

Ces résultats confirment ce que nous avons pu remarquer dans les figures 7 et 8. Les anomalies dans ce jeu de données, obtenues suite à de multiples attaques sur un système de traitement d'eau, ne sont pas saillantes et donc ne sont pas très bien détectées par les algorithmes non supervisés. Cela nous semble logique, car c'est dans l'intérêt des attaquants de rester inaperçus et leur permettre de reproduire les attaques.
Cependant, le succès que les algorithmes supervisés ont eu dans la détection des anomalies nous montre qu'il y a des motifs dans la fluctuation des valeurs des différents attributs qui annoncent ou bien qui montrent qu'une attaque est en train de se produire. Ces motifs sont ensuite appris et des prédictions peuvent être effectuées avec confiance.
Nos résultats nous montrent aussi que le système de traitement d'eau est vulnérable aux nouveaux types d'attaque car nous ne pouvons pas nous reposer sur les résultats des algorithmes de détection d'anomalies (IF et LOF). Pour sécuriser le système, il faudrait déjà avoir accès aux observations faites pendant un type d'attaque pour pouvoir détecter celle-ci par la suite.
Pour améliorer la sécurité de ce système, il serait utile d'étudier les attaques qui ont eu lieu sur d'autres systèmes de traitement d'eau et utiliser des algorithmes supervisés pour apprendre comment les données fluctuent dans ces cas. De cette manière, une fois que ce type d'attaque survient, les algorithmes parviendront à une détection sans délai.