RAPPORT DE STAGE

Machine Learning appliqué à la détection d'anomalies

MALTESE Salomé
23 Juin au 3 Septembre 2021

Tuteurs de stage: M. Precioso Frédéric
Superviseur académique: Mme Malot Christine

Etablissement: Polytech Nice Sophia
Entreprise d'accueil: laboratoire i3S

Sommaire

Remerciements

Je remercie M. Precioso Frédéric et Mme Blay-Fornarino Mireille de m'avoir accueilli et supervisé, ainsi que M. El Amraoui Yassine. Ils m'ont tous les trois aidé et appris des choses tout au long du stage.

Je remercie également Mme Malot Christine, ma tutrice enseignante, Mme Maiffret Julie, gestionnaire des conventions de stage, ainsi que Mme Barrere Sabine, assistance des équipes de recherche du laboratoire pour leur contribution lors de la réalisation de ma convention de stage.

Introduction

Le sujet de ce stage, réalisé au laboratoire i3s est la détection d'anomalies sur des séries temporelles à l'aide d'algorithmes de machine learning ainsi que la capitalisation des expériences effectuées.

Le laboratoire i3s est un laboratoire de recherche en sciences de l'information et de la communication. Il est composé de quatre équipes de recherche. J'ai intégré l'équipe SPARKS (Scalable and Pervasive softwARe and Knowledge Systems) qui est constituée de plusieurs thèmes. Le thème abordé durant le stage est le suivant :
Knowledge Extraction and Learning, qui consiste à développer des méthodes et algorithmes basés sur l'apprentissage automatique (*machine learning*), la fouille de données (*data mining*), l'intelligence artificielle (*artificial intelligence*) pour extraire des données des informations et connaissances**.**
Le but du stage est de faire du machine learning appliqué à un certain problème, la détection d'anomalies sur des séries temporelles, afin d'enrichir une partie de la base de connaissances concernant l'apprentissage automatique.

Dans un premier temps, nous détaillerons le travail demandé. Nous allons notamment définir la détection d'anomalies, le machine learning et comment l'utiliser dans notre contexte. Nous parlerons également de la sauvegarde des connaissances acquises. Ensuite, nous aborderons le travail réalisé, c'est-à-dire comment tracer les expériences et quelles expériences ont été effectuées. Nous aborderons aussi les difficultés rencontrées et si possible comment les surmonter.

Travail proposé

L'objectif de ce stage est de faire de la détection d'anomalies sur des séries temporelles à l'aide de modèles de Machine Learning (ML). La mise en œuvre d'un modèle de machine learning implique plusieurs étapes au-delà de l'exécution de l'algorithme. On appelle le processus comportant toutes les étapes le workflow. Lorsqu'un problème de détection d'anomalies se pose, l'espace de recherche de solution (ML workflows) est vaste. Le but est de réduire cet espace à travers des critères. L'idée est de les déterminer en effectuant différents tests, en explicitant tout le workflow et en comprenant le résultat. C'est-à-dire, savoir expliquer pourquoi un test a marché ou non.

Motivation : détection d'anomalies dans les séries temporelles

La détection d'anomalies est utilisée dans différents domaines d'applications tels que la détection de fraude (fraude pour les cartes de crédit ou fraude à l'assurance par exemple), détection d'incidents liés à la cybersécurité (des intrusions non autorisées et des logiciels malveillants), la détection de panne (de moteurs de voiture par exemple), etc.

Les séries temporelles sont un ensemble de valeurs dont chacune est associée à un temps t. Autrement dit, il s'agit de l'évolution d'une observation au fil du temps. Par exemple, en mesurant la température d'une pièce chaque jour à la même heure, on obtient une série temporelle avec un pas de temps régulier de 24h.

Remarques:

- Nous avons parlé de séries temporelles à une seule variable mais une série temporelle peut aussi être composée de plusieurs variables. On pourrait reprendre l'exemple de la température en ajoutant d'autres variables comme le taux d'humidité par exemple.
- En dehors des séries temporelles, la science des données recouvre de nombreux autres problèmes (computer vision, natural language processing,…)

Définition : détection d'anomalies

La détection d'anomalies correspond à la recherche de valeurs ou ensemble de valeurs dans un jeu de données (dans notre cas une série temporelle) ne correspondant pas au comportement attendu.

Il existe différents types d'anomalies:

- anomalie ponctuelle : une instance définie comme anormale par rapport au reste des données (une "instance" représente ici une observation, composée d'une ou plusieurs variables).

- anomalie contextuelle : une instance définie comme anormale par rapport au reste des données, dans un certain contexte. Une autre instance avec des valeurs similaires peut donc être considérée normale. En reprenant l'exemple vu plus haut: si on mesure cette température pendant un an, une température très basse en hiver serait considérée normale, alors que cette même température en été serait une anomalie à cause du contexte, c'est-à-dire qu'il ne fait pas aussi froid en été.

- anomalie collective : un groupe d'instances anormal par rapport au reste du jeu de données. Par exemple, une période non respectée dans un signal périodique.

source

Méthodologie : Machine Learning et Autoencoders

Le Machine Learning correspond à l'implémentation d'algorithmes qui, lorsqu'on leur donne des données en entrée, vont être capable de prendre des décisions permettant d'arriver au résultat désiré.
Par exemple, dans notre contexte, on donne à notre algorithme de ML une série temporelle et lui nous donne en sortie une classification de chaque observation: soit "normale", soit "anomalie".

Remarque: le résultat désiré n'est pas forcément une classification comme dans notre cas, cela peut être de la prédiction, de l'interprétation de texte, etc.

L'entraînement d'un modèle en ML

Lorsqu'on développe un modèle de ML, il passe par une phase d'apprentissage sur des données appelées les données d'entraînement, dans notre cas une ou des séries temporelles. Le modèle va récupérer sur ces données d'entraînement des descripteurs, c'est-à-dire qu'il va extraire des connaissances sur ces données pour appuyer et améliorer la prise de décision.

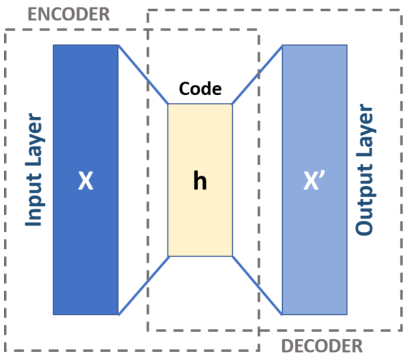
Après cette phase d'entraînement, il y a une phase de test sur un jeu de données avec lequel il ne s'est pas entraîné pour vérifier qu'il est efficace avec un jeu de données indépendant de celui d'entraînement.

Il existe différentes méthodes de ML, ici nous nous concentrons sur la détection "non supervisée". C'est une détection ne comprenant pas de données avec label pour s'entraîner. Dans notre cas, nous cherchons à faire une classification des données avec deux classes "normale" ou "anormale". La détection non supervisée veut dire ici que nous entraînons notre modèle de ML sur des données dont la nature (normale ou non) n'est pas prédéterminée. C'est le scénario le plus largement applicable car nous n'avons pas toujours des données étiquetées en tant que "normales" ou "anormales" dans les problèmes rencontrés réellement.

Il existe plusieurs méthodes non supervisées, durant ce stage nous utiliserons principalement des autoencoders.

<u>Autoencoders</u>

Un autoencoder est un réseau de neurones qui encode les données en une représentation dans un espace de dimension plus faible appelé espace latent (compression des données) et qui les décode la représentation latente des données d'entrée pour les reconstruire.



X : vecteur avec les données d'entrée

X' : vecteur de même dimension que X avec les valeurs reconstruites

h : espace latent

Schéma d'autoencoder basique

source

Les autoencoders reconstruisent les données d'entrées approximativement, en préservant uniquement les aspects les plus pertinents des données. Nous supposons alors dans le contexte de détection d'anomalies, que l'autoencoder encode les parties normales du jeu de données correctement et de façon erroné les anomalies. Ainsi, nous pouvons calculer une erreur de reconstruction entre les vecteurs X' (jeu de données reconstruit) et X (jeu de données d'entrée). Une erreur plus élevée indique des anomalies.

Objectif final : capitalisation des expériences

Motivations

Comme nous l'avons vu plus haut, lorsqu'un problème nécessitant de l'apprentissage automatisé se pose (que se soit de la détection d'anomalies ou non), l'ensemble de solutions possibles est vaste. Cependant, toutes les solutions ne donnent pas les mêmes résultats. L'idée en traçant les expériences effectuées est de trouver quelles sont les solutions les mieux adaptées au problème initial et les généraliser à ce type de problème. L'ensemble des solutions est alors restreint, et normalement, la quantité de travail aussi lorsqu'un prochain problème du même type se pose (ce type est à définir, déterminer quelles conditions font que les solutions marchent dans ce cas là).

Restreindre les possibilités

Lorsqu'on parle d'un ensemble de solutions possibles, cela correspond en ML à un ensemble de workflows possibles. C'est-à-dire, l'ensemble du processus de traitement d'un jeu de données, qui comprend le choix et le paramétrage de l'algorithme de ML, ainsi que le nettoyage des données (détection et correction/suppression d'erreurs présentes sur des données), l'extraction de features, les preprocessing utilisés, les méthodes d'évaluation, les métriques utilisées pour valider un modèle.

Pour pouvoir restreindre les possibilités, il faut détailler l'ensemble du workflow et expliquer le plus possible ses choix et les résultats obtenus, notamment lorsqu'ils sont mauvais.

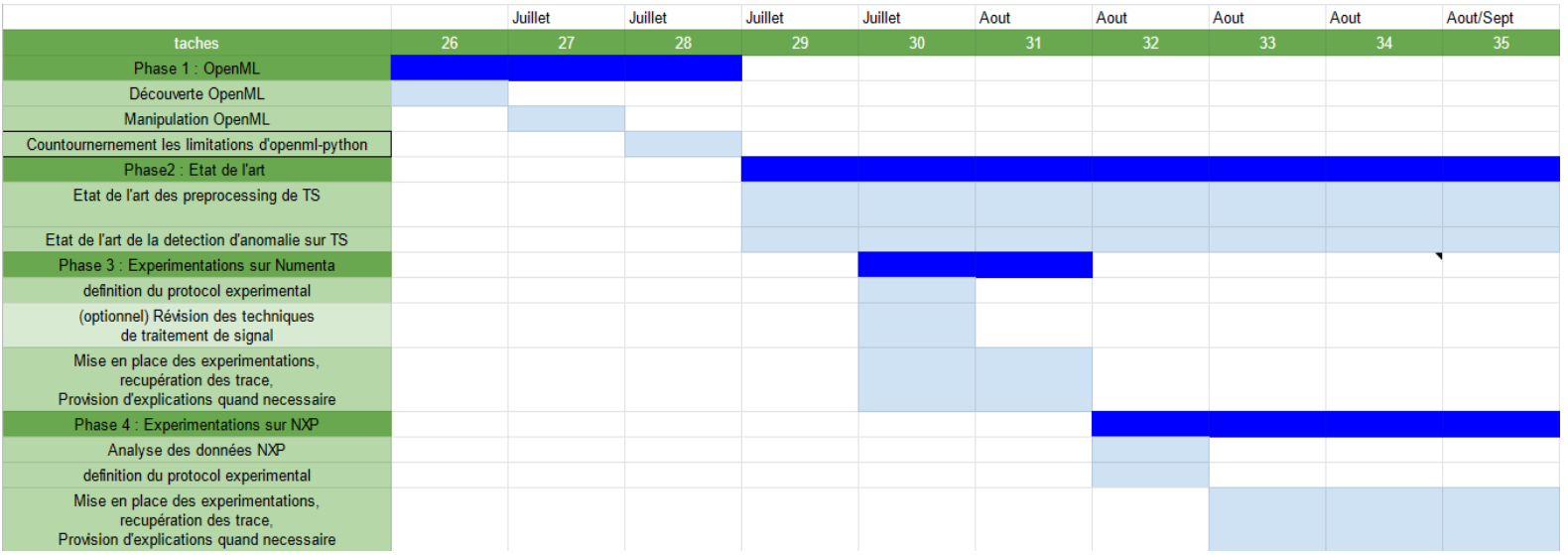
Difficultés:

- Les domaines concernés par la détection d'anomalies sont très variés et la manière de les résoudre diffère souvent.

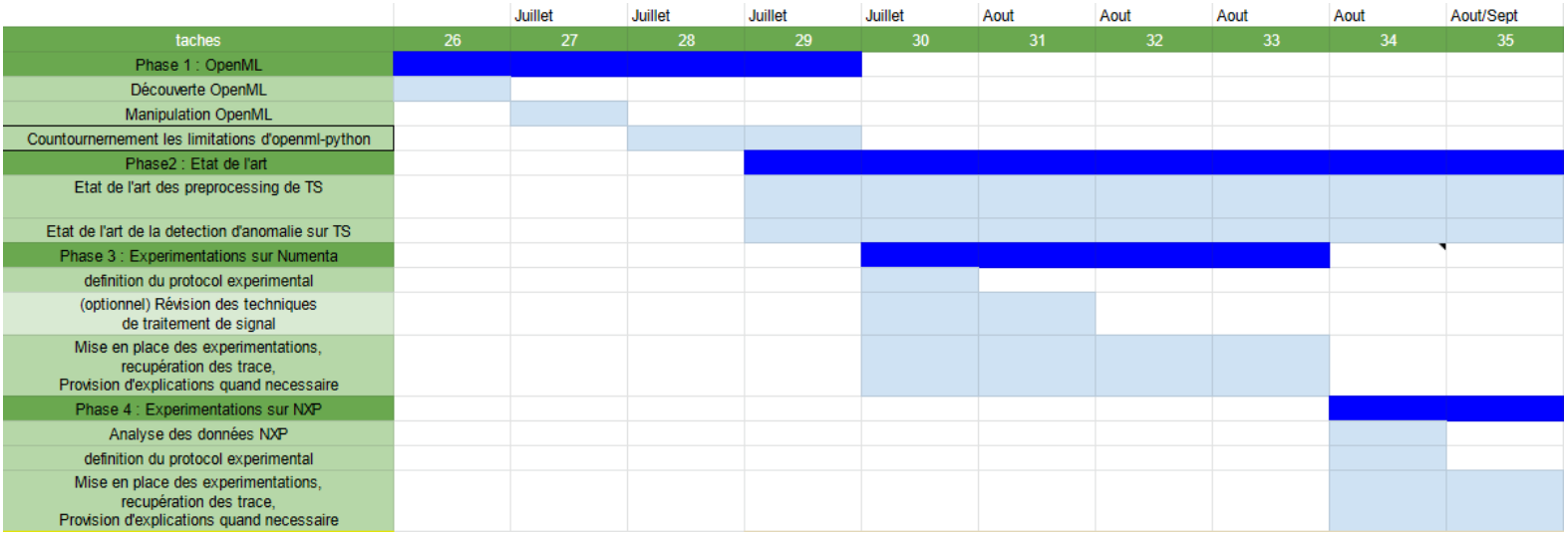
- Expliquer pourquoi une méthode ne marche pas n'est pas toujours évident, mais cela apporte des informations intéressantes à la base de connaissances

Travail réalisé

planning prévisionnel:



planning réel:



OpenML

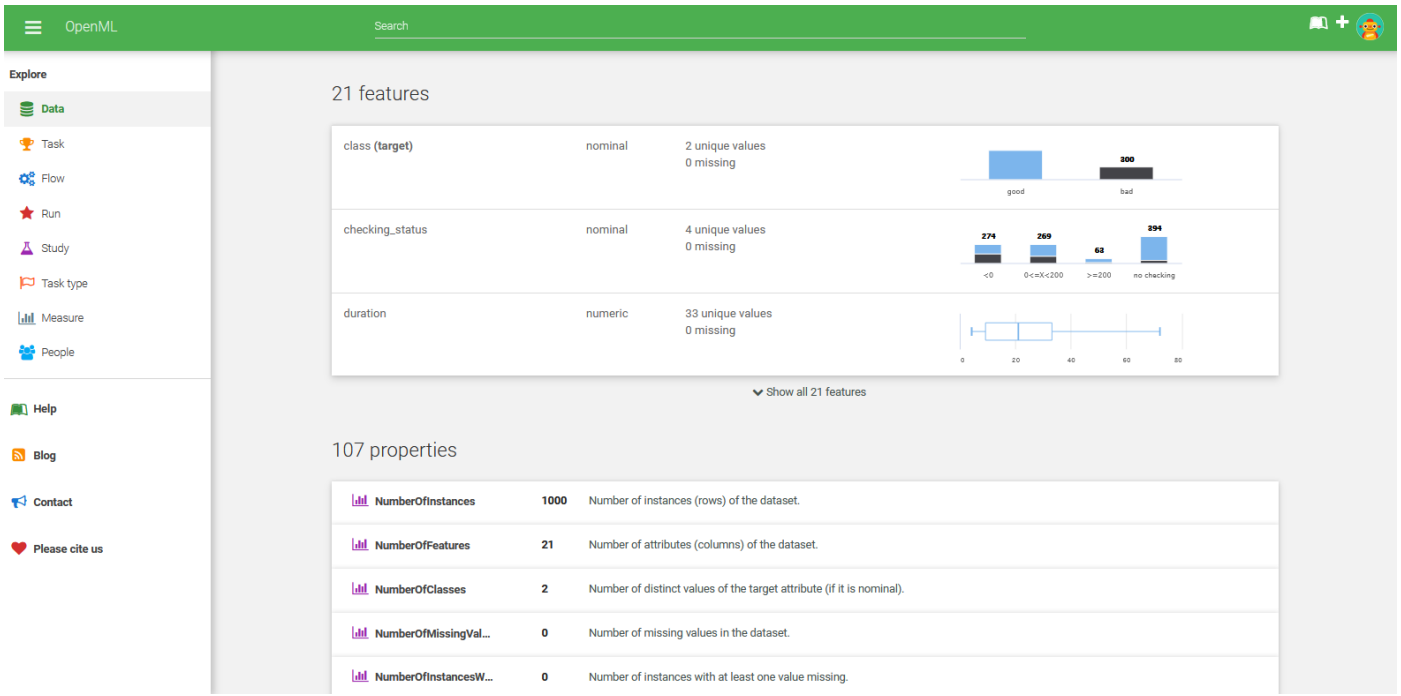
OpenML est une plateforme en ligne permettant de partager des jeux de données, des algorithmes de ML et des expérimentations.
Il existe une interface Python pour OpenML, permettant de faire les expérimentations en Python (avec les extensions scikit-learn et tensorflow, librairies Python pour le ML).
Lien vers la plateforme : https://www.openml.org/

Les composants d'OpenML:

- Datasets: les jeux de données. OpenML fournit également un certain nombre de propriétés comme par exemple le nombre d'instances ou le nombre de valeurs manquantes.

- Tasks : une tâche de machine learning (classification supervisée par exemple), associée à un jeu de données.
- Flows: algorithmes ou scripts résolvant des tâches. Le flow contient toutes les informations pour appliquer un certain algorithme à une tâche.
- Runs: applications de flows sur des tâches spécifiques

source



Présentation d'un jeu de données sur OpenML

Utilisation d'OpenML dans le contexte du stage

Le formalisme d'OpenML, notamment sa façon de tracer les flows et les runs, est intéressant. Cela permet de tracer les expérimentations (les algorithmes utilisés mais aussi les paramètres choisis), ce que nous voulons faire pour essayer de comprendre pourquoi certains paramètres, certaines méthodes sont plus adaptées que d'autres. Cette façon de sauvegarder pourrait nous aider à associer certaines méthodes de pré-traitement des données, certains algorithmes de machine learning à un certain type de problème initial.

Difficultés rencontrées

Il est possible de construire des modèles de ML / des pipelines (modèle + pré-traitement des données) et de générer des flows et des runs grâce aux fonctionnalités d'OpenML. Cependant, nous perdons la main sur certaines actions de l'entraînement du modèle qui sont faites directement par la plateforme, ce qui peut être contraignant. Comme mentionné plus haut, l'interface Python possède deux extensions, mais si nous voulons faire un pré-traitement des données sans l'utilisation de ses extensions ou plus généralement un workflow customisé qui n'utilise pas que des méthodes de ses deux librairies, générer un run ou un flow n'est pas toujours possible

De plus, ce n'est pas toujours évident de corriger des erreurs de code lorsqu'elles sont liées au fonctionnement d'OpenML. En effet, il n'y a pas forcément une grande communauté donc souvent nous ne trouverons pas de personnes ayant déjà rencontré et partagé ce problème avec éventuellement des solutions proposées par d'autres personnes. Tout est bien documenté pour s'aider, mais c'est compliqué si il y a des points de la documentation mal compris.

Contournement des limitations d'OpenML

OpenML ne limite pas qu'à la traçabilité des expérimentations, mais c'est cet aspect qui nous intéresse, notamment l'enregistrement local des flows et des runs. La sauvegarde du flow peut facilement passer par OpenML si nous construisons nos modèles à partir des extensions. Si ce n'est pas le cas, en s'inspirant de la manière dont les flows/runs sont sauvegardés par OpenML, nous pouvons finalement faire les sauvegardes nous-même. Cela apporte même une certaine flexibilité puisque nous pouvons choisir d'enlever des informations qui ne nous intéressent pas forcément et/ou d'en ajouter.

Expérimentations sur Numenta Anomaly Benchmark

Le Numenta Anomaly Benchmark (NAB) est une référence pour l'évaluation des algorithmes de détection d'anomalies. Il comprend plusieurs jeux de données correspondant à des séries temporelles réelles ou artificielles. source

Protocol experimental:

- sélection des jeux d'entraînement et de test
- pré-traitement des données
- choix de l'algorithme de ML
- évaluation

Les métriques utilisées pour l'évaluation sont la précision, le recall et le f1 score (définis en Annexe C). Après évaluation, analyser les résultats et valider ou non le workflow en fournissant des explications.

Le but est de se familiariser avec certains algorithmes de ML, certaines méthodes de pré-traitement des données et de tracer les expériences, analyser les résultats afin de déterminer ce qui fonctionne le mieux selon le type de données d'entrée, le problème, … Pour cela, nous allons réaliser plusieurs expérimentations testant différentes méthodes et algorithmes.

Test de différentes méthodes de pré-traitement des données

Lorsque nous travaillons avec des séries temporelles, les possibilités de pré-traitement des données sont nombreuses.

Il y a les pré-traitements classiques qui sont utilisés également à d'autres types de jeux de données. Par exemple, la normalisation des données, permettant s'il y a plusieurs attributs qu'ils aient tous le même "poids" pour l'algorithme (un attribut avec un plus grand intervalle de valeurs pourrait être dominant par rapport aux autres, ce que nous pouvons éviter en mettant tout à la même échelle). C'est aussi nécessaire avec certains algorithmes, comme par exemple les réseaux de neurones qui convergent plus lentement (ou pas du tout) sans passer par cette étape.

Il y aussi des transformations de données propres aux séries temporelles.Il existe par exemple:

- la décomposition de la série en plusieurs éléments : tendance, saisonnalité et résidu. Au lieu d'entraîner notre modèle sur l'attribut "brut", nous pouvons l'entraîner sur ceux obtenus après décomposition qui peuvent apporter des informations plus intéressantes au modèle.
- la transformée de Fourier : passage du signal dans le domaine temporel au domaine fréquentiel. Ceci peut apporter de nouvelles informations sur la série temporelle ou permettre de filtrer le signal. Par exemple, nous pouvons supprimer les hautes fréquences de notre signal et faire ensuite une transformée de Fourier inverse (remettre le signal dans le domaine temporel), permettant d'enlever le bruit.

Test de différents algorithmes de ML

Comme nous l'avons vu à la page 6, nous utilisons des autoencoders. Nous pouvons faire varier l'architecture de l'autoencoder en utilisant différents types de réseaux de neurones. Parmi les différentes architectures testées, nous avons par exemple:

- les réseaux de neurones convolutifs (CNN) : souvent utilisés pour des données spatiales comme les images par exemple. Cependant, ils peuvent être utilisés avec des séries temporelles. Si par exemple, il n'y a pas de dépendance par rapport au temps. c'est a dire si une variable n'est pas corrélée avec elle même a un temps précédent, le CNN peut être pertinent.

- les réseaux de neurones récurrents (RNN) : usage pour données séquentielles. Ils prennent en compte la dimension temporelle. Dans le contexte de détection d'anomalies, si par exemple

  une anomalie est toujours précédée d'un certain comportement, les RNN sont capables de le prendre en compte et peuvent donc être plus intéressant qu'un CNN. Nous utiliserons les LSTM (voir annexe A) qui sont une variante des RNN.

Processus d'expérimentation avec un exemple:

Sélection des jeux d'entraînement et de test: trois jeux de données du NAB. Les jeux de données sont constitués d'un seul attribut : "valeur" (ce sont des données générées artificiellement, l'attribut n'a pas de signification particulière).

Figure 2: jeu d'entraînement sans anomalies

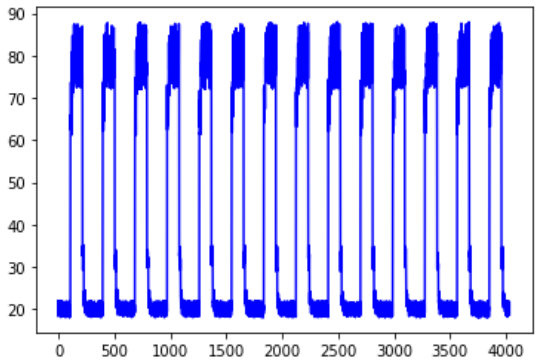


Figure 3: 1er jeu de test avec anomalies

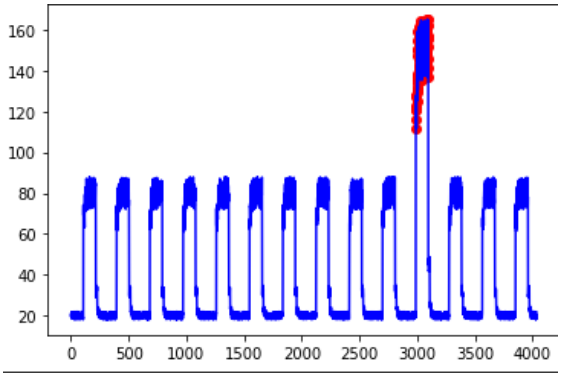
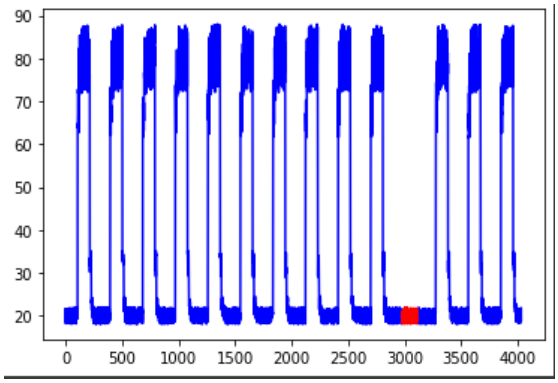


Figure 4: 2nd jeu de test avec anomalies



Observations:

Les anomalies des jeux de données de test sont représentées en rouge. Dans les deux cas, en se référant aux différents types d'anomalies définis plus tôt, on pourrait définir l'ensemble des points

rouges comme une anomalie collective puisque la périodicité n'est plus respectée. Cependant, les deux cas sont différents : dans le premier, les anomalies sont des valeurs n'entrant pas dans l'intervalle du jeu de données d'entraînement. On pourrait donc aussi définir chaque point rouge un à un comme une anomalie ponctuelle. Ce n'est pas le cas du second, les valeurs sont comprises dans l'intervalle du jeu sans anomalies. Cela signifie que ces points ne sont pas anormaux dû uniquement à leur valeur mais plutôt au fait qu'ils aient cette valeur à un instant t.

Remarque: ici les données sont labellisées, mais les labels ne seront pas utilisés durant la phase d'entraînement de l'autoencoder. Nous utiliserons l'erreur de reconstruction du jeu de données d'entraînement pour évaluer le modèle. Comme il n'y a pas d'anomalies, l'autoencoder est censé bien reconstruire le jeu de données et ainsi générer une erreur de reconstruction faible. Nous testerons ensuite notre autoencoder entraîné sur les jeux de données de test. Dans ce cas, nous nous attendons à une erreur plus élevée dans les zones rouges. Il faut ensuite définir un seuil tel que chaque point avec une erreur de reconstruction supérieure à ce seuil soit une anomalie.

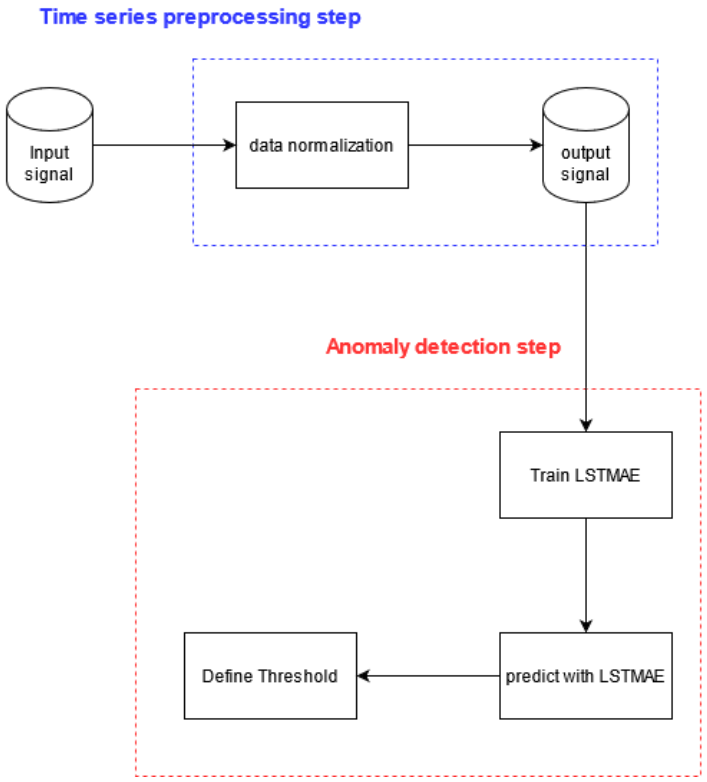


Schéma du workflow utilisé

Définition du seuil : annexe B

Les sauvegardes pour tracer l'expérience:

- Flow et run: algorithmes utilisés ainsi que leurs paramètres pour le pré-traitement des données et pour l'autoencoder, dont chaque couche du réseau de neurones est détaillée.
- Prédictions : valeurs reconstruites par l'autoencoder
- Métriques : pour mesurer l'efficacité du workflow (uniquement lorsque nous avons des labels).

Le run et le flow sont sauvegardés dans des fichiers json. Les prédictions sont sauvegardées dans un tableau, où l'on y ajoute les valeurs réelles du jeu de données pour faciliter l'évaluation, si besoin d'être calculée plus tard. Nous pouvons ensuite analyser les résultats.

Dans ce cas, ils étaient corrects avec le premier jeu de test, mais pas avec le second. En associant cette expérience et ces résultats aux observations faites ci-dessus, nous obtenons des informations permettant de compléter notre base de connaissance. Nous testons ensuite d'autres workflows pour le cas non résolu.

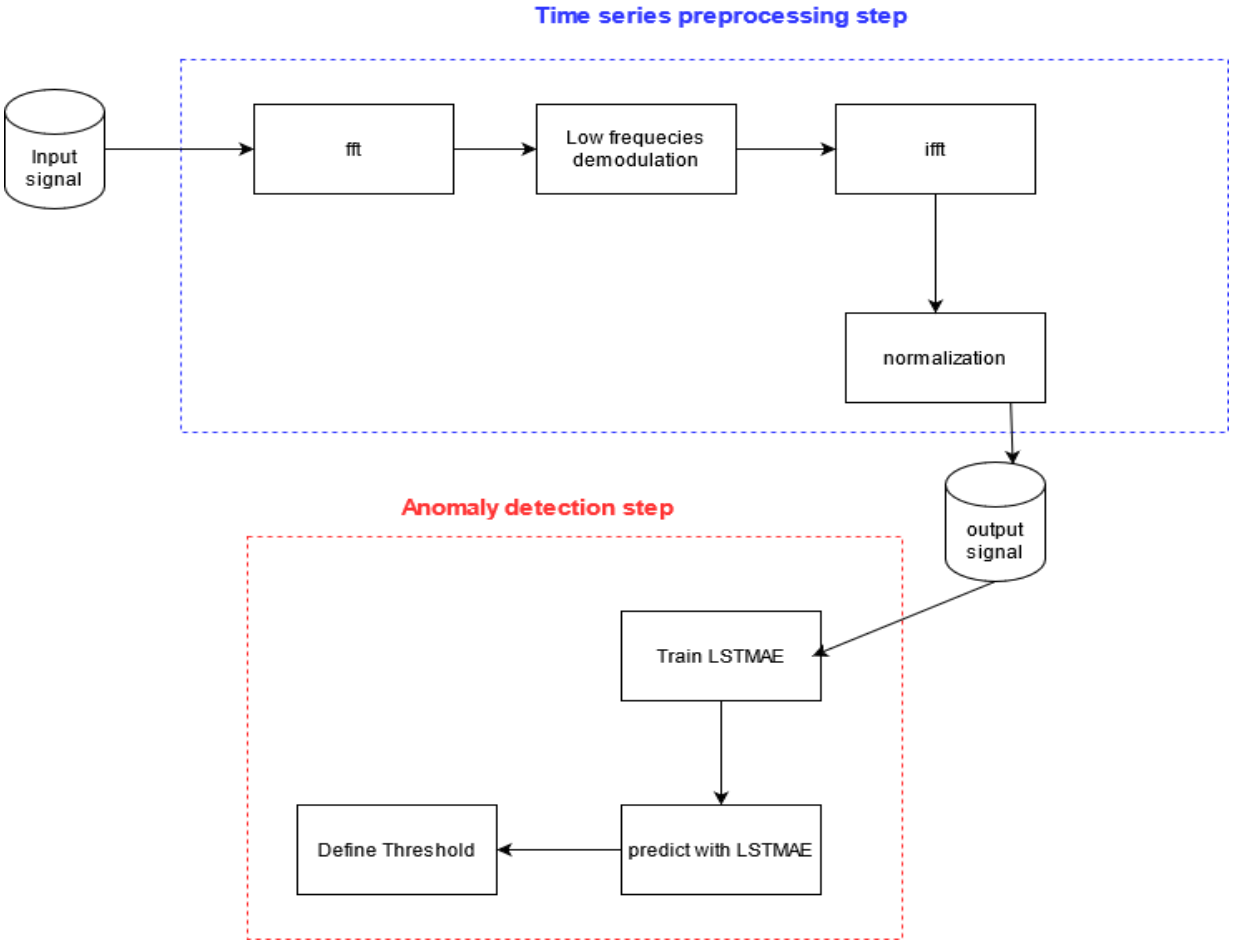


Schéma du nouveau workflow

fft : Fast Fourier Transform, c'est un algorithme de calcul de la transformation de Fourier discrète (TFD) avec une complexité moindre que l'algorithme classique.
ifft : Inverse Fast Fourier Transform, correspondant à l'algorithme de calcul de la transformation de Fourier inverse.

En supprimant les amplitudes correspondant aux basses fréquences de notre signal fréquentiel, nous avons enlevé de l'information sur la partie cyclique de notre série temporelle, partie que l'entraînement et le test ont en commun. Cette méthode a permis de mettre en avant les anomalies du jeu de test, que notre autoencoder a pu détecter.

Expérimentation sur jeu de données réelles

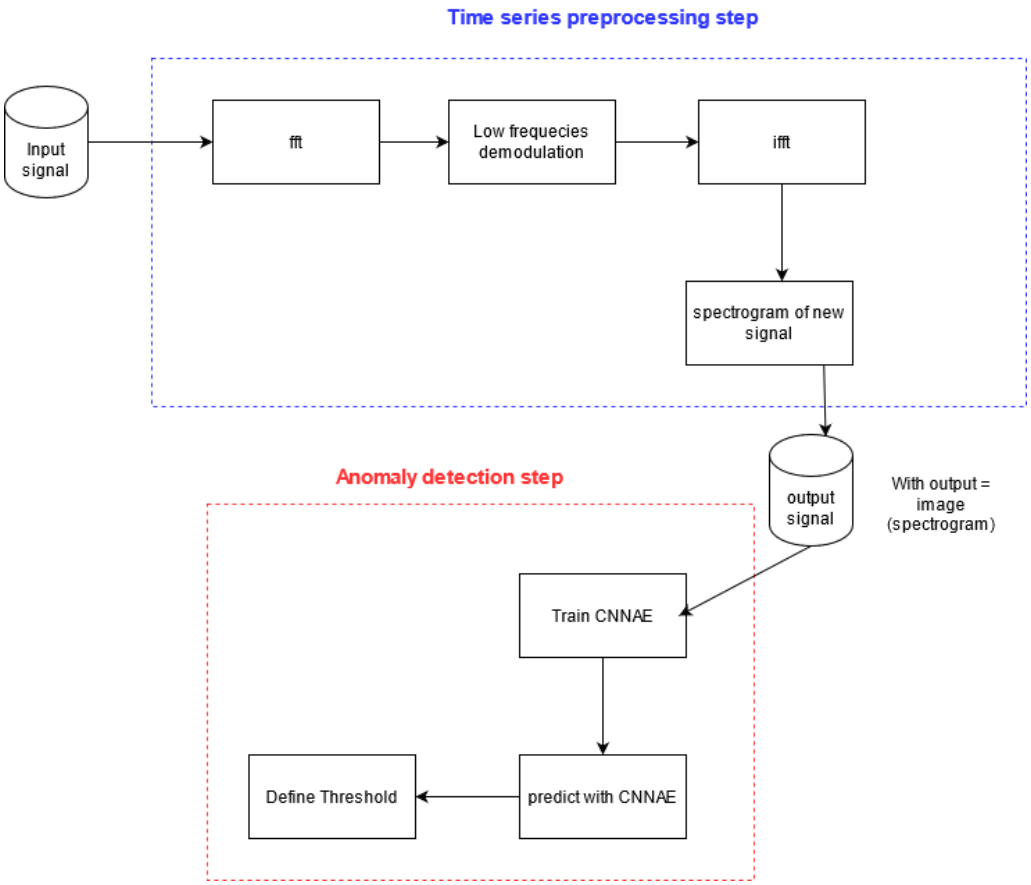
Le problème consiste à faire de la détection d'anomalies dans les roulements d'un moteur. Pour cela, le courant a été mesuré à la surface du roulement puis traduit en voltage. Plusieurs expériences ont été effectuées avec différentes caractéristiques (variation de la vitesse du moteur par exemple). A chaque fois, les expériences sont faites sur un roulement normal et un roulement avec un trou. Nous avons alors des jeux de données entièrement normaux ou anormaux (contrairement aux exemples vu dans la partie précédente où les anomalies ne représentaient qu'une partie du jeu de données).

Les données sont des signaux périodiques. Les données normales et anormales bruts ne semblent pas différenciables : même distribution avec mêmes statistiques (moyenne, écart-type, maximum). L'idée est alors de mettre en avant grâce à des pré-traitements de données ce qui caractérise les jeux de données comme normaux ou non. Il faut essayer de trouver comment nous définissons une anomalie dans ce contexte.

Idées de pré-traitement des données:

- celui du WF 2 de la partie précédente (avec la transformée de Fourier).
- décomposition de la série temporelle en tendance, saisonnalité et résidu. La tendance et la saisonnalité des séries normales et anormales seront similaires mais l'idée est de voir s' il y a des différences au niveau du résidu, qui permettrait à l'autoencoder de les différencier.

Ces pré-traitements de données n'ont pas permis de bien différencier une série normale d'une avec anomalies. Par manque de temps, je n'ai pas encore trouvé une solution à ce problème. Une idée de prochain test était d'approfondir le pré-traitement des données avec la transformée de Fourier et de travailler avec des images, qui sont souvent traitées avec les CNN.



Idée du workflow en schéma

Conclusion

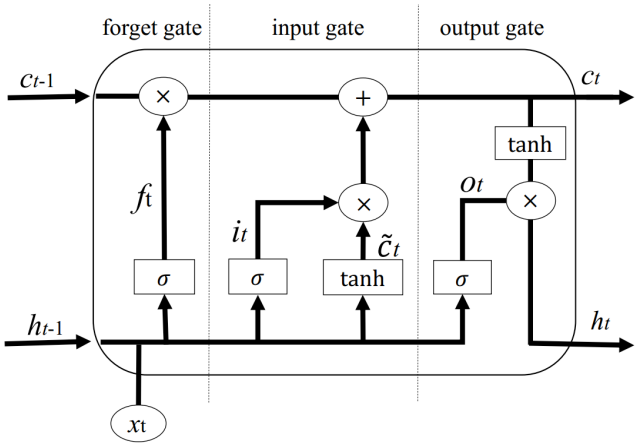
Tout au long du stage, j'ai acquis des connaissances. Au départ, plutôt accès sur la traçabilité des expériences avec la manipulation d'OpenML où il fallait comprendre ce que nous voulons sauvegarder et comment le faire. Ensuite, j'ai découvert un certain nombre de méthodes de pré-traitement de données, notamment celles spécifiques aux séries temporelles. Je n'ai pas pu citer tout ce que j'ai testé dans ce rapport et je n'ai pas pu testé toutes les méthodes possibles durant le stage. C'est une des raisons de la difficulté à restreindre l'espace des solutions, qui était notre objectif de départ. Il existe énormément de possibilités. Je me suis également familiariser avec les autoencoders et certains types de réseaux de neurones. En testant et comparant différentes architectures d'autoencoders, avec différents problèmes initiaux, j'ai pu avoir une idée de quelle architecture est mieux et dans quel cas. Cependant, la comparaison est faite dans un cercle restreint puisqu'il n'y pas que les autoencoders qui permettent de détecter des anomalies. De plus, les cas testés ne représentent pas tous les problèmes possibles. J'ai pu m'en rendre compte en travaillant sur un jeu de données réel où les anomalies étaient plus difficiles à caractériser. Les workflows plutôt simples, c'est-à-dire sans trop d'étapes de pré-traitement des données, n'ont pas permis au modèle de détecter des anomalies. Cependant, les connaissances acquises ont permis de trouver des pistes à approfondir.

En traçant nos expérimentations sur une période plus longue que ce stage et sur plus de problèmes, il est à mon avis possible de restreindre de manière significative ce vaste espace de solutions. Cet espace restreint serait bénéfique à la fois pour les personnes n'ayant pas l'expertise du Data Scientist mais également pour le Data Scientist lui-même. Car en posant des explications sur nos choix, nous prenons plus de recul sur ce que nous faisons et cela amène à encore plus de compréhension sur le fonctionnement des méthodes, des algorithmes, de tous les paramètres utilisés, etc… Cela apporte aussi plus de réflexions, ce qui peut conduire à de nouvelles idées de résolutions.

Annexe

A. LSTM Networks

Les réseaux de neurones LSTM (Long Short Term Memory) sont un type de réseaux de neurones récurrents capable d'apprendre des dépendances long termes, c'est-à-dire capable de garder de l'information sur une période de temps longue.



Cellule d'un réseau de neurones LSTM, source: 5

Le forget gate détermine quelle information de la cellule précédente nous voulons oublier. L'input gate correspond à l'information que nous voulons garder de la cellule actuelle. L'output gate correspond à l'information que nous allons envoyer au prochain hidden state (ht).

B. Définition d'un seuil

On calcule l'erreur de reconstruction : E = | x_train - x_train_prediction |

avec x_train : valeurs du jeu d'entraînement et x_train_prediction : valeurs reconstruites par l'autoencoder

1. Seuil = max(E)
Ainsi, on définit une erreur sur le jeu de test plus grande que toutes les erreurs calculées sur le jeu d'entraînement comme une anomalie.

2. Seuil = mean(E) + z * std(E)

Ce calcul de seuil apporte une certaine flexibilité grâce à la variable z qui nous permet de décider à quel point nous voulons prendre en compte l'écart type

Remarque: nous pouvons aussi travailler sur des séquences et calculer une erreur par séquence. Nous choisissons ensuite un seuil tel que: seuil = argmax(mean(E) + z * std(E)).

C. Métriques

$$\text{Precision} = \frac{tp}{tp + fp}$$
$$\text{Recall} = \frac{tp}{tp + fn}$$

tp = true positive. Dans notre contexte, une anomalie détectée correctement.
fp = false positive. Dans notre contexte, une anomalie détectée par erreur.
fn = false negative. Dans notre contexte, une anomalie non détectée.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Le F score correspond à la moyenne arithmétique entre la précision et le recall. Plus il est élevé, plus le modèle est performant.

Bibliographie

1. Varun Chandola, Arindam Banerjee & Vipin Kumar. Anomaly detection : A survey. July 2009, ACM Computing Surveys

2. https://en.wikipedia.org/wiki/Autoencoder

3. https://docs.openml.org/

4. Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, Frank Hutter, *OpenML-Python: an extensible Python API for OpenML,* Journal of Machine Learning Research 22(100), 2021, https://arxiv.org/abs/1911.02490

5. Pan, H.; He, X.; Tang, S.; Meng, F. An improved bearing fault diagnosis method using one-dimensional CNN and LSTM. 2018

6. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding Hundman et al., *KDD2018*

7. https://en.wikipedia.org/wiki/Precision_and_recall

8. Andrew Cook, Goksel Mısırlı, and Zhong Fan*, Anomaly Detection for IoT Time-Series Data: A Survey,* IEEE Internet of Things Journal, décembre 2019

9. Seif-Eddine Benkabou. *Détection d'anomalies dans les séries temporelles : application aux masses de données sur les pneumatiques*. Université de Lyon, 2018

10. Daniel Morinigo-Sotelo, Rene Romero-Troncoso and Joan Pons-Llinares, *Current-Based Bearing Fault Diagnosis Using Deep Learning Algorithms,* Energies, 2021

11. https://github.com/numenta/NAB/blob/master/README.md