TP: Détection d'anomalies avec des réseaux de neurones

Alexandre Dey

2 février 2021

1 Introduction

L'objectif du TD est d'implémenter un réseau de neurone auto-encodeur et de trouver une configuration optimisée afin de répondre à un problème de détection d'anomalie.

Le jeu de données fourni est extrait du dataset UNSW-NB15 [1]. Ce dataset orienté cybersécurité regroupe plusieurs attributs relatives à des connexions réseau au sein d'un système d'information. Certaines de ces connexions sont attribuables à des attaques, d'autres à du trafic normal.

Le nombre d'attaques labellisées étant très faible, nous sommes dans un contexte d'apprentissage non-supervisé, et plus spécifiquement de détection d'anomalies (i.e on apprend sur les connexions normales et on considérera les anomalies comme étant de potentielles attaques).

Note : Le code doit être inséré au niveau des TODO dans le base code.

2 Exercice 1 : Préparer la donnée

L'essentiel du code pour cet exercie est à remplir dans le fichier `1-prepare.py`. Il faudra également compléter les fichiers dans le répertoire `libTP/feature_engineering/`.

2.1 Identifier les types des variables

Le dataset contient plusieurs variables de différents types (catégorielle, binaire et numériques) et chacun de ces types doit être géré différemment des autres. Identifier le type de chacune des variables.

1. Source : `https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/`

2.2 Transformation des variables

Chacune des variables doit être transformée en nombres avant de pouvoir être utilisée. Cette méthode de transformation est différente en fonction du type de variables.

Les fichiers à modifier se trouvent dans `libTP/feature_engineering/`.

2.3 Préparer la structure du modèle

Le modèle qui sera implémenté au cours de ce TP commence par appliquer un fonction de transformation différente (ou *embedding* en anglais) à chacun des attributs d'entrée. Pour pouvoir créer ces fonctions de transformation, pour chaque attribut, il faudra donner au minimum :

— Le type de l'attribut (`categorical`, `binary`, `numerical`)

— La dimension de sortie `output_size` (un entier)

Dans le cas des variables catégorielles, il faudra également fournir maximum de catégories différentes. Cette valeur peut être trouvée grâce au membre `max_size` de la class `CategoricalFE`.

2.4 Sauvegarder sur disque

Il est impératif de bien sauvegarder les configurations faites pendant cet exercice. En effet, elles seront réutilisées tout au long du TP pour construire le modèle ainsi que transformer la donnée de manière identique à chaque fois. Pour celà, vous pouvez employer la méthode `save` déjà impléntée pour les transformateurs de donnée. Pour la configuration générale, vous pouvez employer le module `json` de la librairie standard de python.

3 Exercice 2 : Entraîner l'auto-encodeur

3.1 Implémenter les briques de l'auto-encodeur

L'essentiel du modèle est déjà fourni dans le répertoire `libTP/models/`. Il faut cependant implémenter les fonctions d'encodage et de décodage pour chacun des attributs. Chaque type d'attribut va également nécessiter une fonction coût différente. Il est recommandé de chercher dans la documentation de `pytorch` pour cette étape.

3.2 Entrainer le réseau

Une fois toutes les briques de l'auto-encodeur implémentées, il est temps d'entraîner le premier modèle. Une grosse partie de la boucle d'entraînement est abstraite par `pytorch-lightning`. Pour implémenter l'*early stopping*, vous aurez besoin de faire appel à des *callbacks* fournis par `pytorch-lightning` : `ModelCheckpoint` et `EarlyStopping`.

3.3 Calibrer le calul du score d'anomalie

Si vous lancez le modèle tel quel, vous pouvez remarquer qu'en fonction de l'attribut, l'erreur de reproduction calculée peut varier différement (ex : grandes ou petites valeurs). Pour éviter que le score d'anomalie total ne soit trop influencé par les attributs ayant des *loss* en moyenne plus forte au détriments de celles plus faibles, il faut remettre à l'échelle ces valeurs.

Pour ce faire il est par exemple possible d'appliquer une standardisation (soustraire la valeure moyenne puis diviser par la variance). Il faut l'implémenter dans deux méthodes de la classe `AutoEncoder` :

— `calibrate` où il faudra, pour chaque attribut différent calculer la moyenne et la variance

— `score` où il faudra soustraire la moyenne et diviser par la variance

Il s'agira ensuite d'appeler la méthode `calibrate` sur les données de test une fois le réseau entraîné et de sauvegarder le modèle et les paramètres utiliser pour le calcul du score d'anomalie.

Note : La standardisation retournera des valeurs positives et négatives. Les valeures négatives seront celles qui seront très "normales" (leur erreur de reproduction étant en dessous de la moyenn). Pour éviter qu'elles ne diminuent le score des anomalies, il et possible d'utiliser la fonction `torch.clip()` pour borner les valeures (ex : remettre à zéro les valeurs négatives).

4 Exercice 3 : Définir un seuil de décision

Commencez par recharger le modèle précédemment entraîné.

Utiliser les méthodes d'évaluation appropriées (cf. cours sur les métriques d'évaluation) pour définir le seuil de décision au delà duquel une connexion sera considérée comme anormale. Inclure les visualisations dans le rapport (vous pouvez utiliser *matplotlib*[2] pour créer les visualisations).

Note : Pour cette étape, vous disposez du dataset *evaluation.csv* pour lequel les attaques sont identifiables par leur label (1 pour les attaques, 0 sinon).

5 Exercice 4 : Labelliser les données inconnues

Après avoir construit un réseau dont les performances sont satisfaisantes, donnez une prédiction pour le label des données du dataset *unknown.csv*. Joindre au rapport un CSV avec trois colonnes :

1. score : le score d'anomalie en sortie de l'autoencodeur
2. pred_label : le label prédit (1 anomalie, 0 normal)
3. hidden_label : fournit dans le jeux d'entrée

2. Voir : `https://matplotlib.org/`