Projet BDA: Anomaly Detection in Network Traffic with K-means Clustering

Authors: Hochet Guillaume, Kopp Olivier, Silvestri Romain

Introduction

Dans le cadre du cours Big Data Analytics, un projet d'apprentissage est réalisé. La technologie utilisée pour sa réalisation est SparkML. Ce document présente la description du projet ainsi que les résultats obtenus.

Partie 1 : Description du dataset et des features

Le dataset contient un ensemble de 4,9 millions de connexions réseau, et a été généré par rapport à des données de 1999. Les informations contenues ont été prétraitées, afin d'avoir uniquement un résumé de celle-ci (un paquet réseau peut avoir une structure complexe et pas forcément toujours uniforme) afin d'obtenir des données facilement utilisables dans un contexte de machine learning. Au final, la taille totale du dataset est de 708 MB.
Chaque entrée du dataset est représentée au format csv, et contient 41 features, comme par exemple les protocoles des différentes couches OSI (tcp/udp, http, ftp, …), ou encore la quantité de données échangées. La plupart des features correspondent à des compteurs, ou à une valeur binaire (0 ou 1). Il existe également certaines features correspondant à un ratio, et donc représenté avec des valeurs décimales comprises entre 0 et 1.
Enfin, la dernière feature est un label, catégorisant le type d'attaque correspondant à la connexion. Dans notre cas, l'apprentissage est non supervisé, et donc cette valeur sera très certainement ignorée.

Une description plus détaillée des features est présentée dans un document en ligne[1] :

feature name	*description*	*type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
flag	normal or error status of the connection	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous

[1] http://kdd.ics.uci.edu/databases/kddcup99/task.html

land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete
count	number of connections to the same host as the current connection in the past two seconds	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
serror_rate	% of connections that have ``SYN'' errors	continuous
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

dst_host_count	Count of connection having same dest hot	continuous
dst_host_srv_count	Count of connection having the same destination host and using same service	continuous
dst_host_same_srv_rate	% of connection having the same destination host and using same service	continuous
dst_host_diff_srv_rate	% of different service on the current host	continuous
dst_host_same_src_port_rate	% of connection to the current host having same src port	continuous
dst_host_srv_diff_host_rate	% of connection to the same service coming form different host	continuous
dst_host_serror_rate	% of connection to the current host that have ``SYN'' error	continuous
dst_host_srv_serror_rate	% of connection to the current host and specified service that have ``SYN'' error	continuous
dst_host_rerror_rate	% of connection to the current host that have ``REJ'' error	continuous
dst_host_srv_rerror_rate	% of connection to the current host and specified service that have ``REJ'' error	continuous

Voici enfin un exemple de connexions normale :
0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.
Ici on a l'exemple d'une connexion http, avec 181 bytes envoyé (la requête GET/POST) et 5450 (le contenu de la page). On peut aussi voir que les pourcentages d'erreurs valent 0.

Puis une connexion catégorisée comme une attaque (ici une attaque de type smurf, une variante de DDOS) :
0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.
Dans ce cas le protocole est icmp, et il y a 0 byte dans la réponse (le principe de l'attaque smurf est d'envoyer un ping avec l'ip de la victime comme source, afin que le serveur réponde directement vers la victime). On peut voir ici un grand nombre de connexions au même port (255 connexions), qui nous montre déjà une certaine anomalie (la requête précédente n'avait que 9 connexions).

Partie 3 : Questions d'analyse

Cette première approche du projet soulève certaines questions, dont nous espérons pouvoir obtenir des réponses à la fin de ce projet.

Est-ce que l'utilisation de toutes les features disponibles donne de meilleurs résultats ? Quel ensemble de features donnent les meilleurs résultats ? Est-ce que certaines ne sont pas nécessaires au clustering ?

Nous n'avons pas réussi à trouver de méthode efficace existante pour la feature selection dans un algorithme K-means. Les différentes méthodes proposées par Spark (*RFormula, VectorSlicer,...*) ne répondaient pas à nos attentes au niveau du traitement de nos features et des subset retournés. Nous avons donc voulu utiliser une méthode naïve pour commencer. La méthode consiste à retirer une feature, puis comparer le score obtenu avec celui prenant en compte toutes les features. Si le résultat est meilleur, la features est retirée, sinon, on la conserve. Le problème rencontré concerne le temps de calcul nécessaire à une méthode comme ça. En effet, la convergence de l'algorithme K-means dépend de l'emplacement de départ des centroïdes, qui est aléatoire. Pour obtenir des résultats corrects, il devient donc nécessaire d'exécuter l'algorithme un certain nombre de fois, puis de moyenner les résultats. Avec 42 features et un temps moyen d'exécution de 15min par K-means, cela devient très rapidement trop long (le déploiement sur un cluster à été testé, mais sans succès, à cause de problèmes de version sur spark notamment).

Nous avons donc décidé d'opter pour une autre méthode. Nous avons recherché dans les différentes études déjà effectuées par le passé sur ce dataset et avons trouvé plusieurs sous ensemble de features qui avaient été sélectionnés par d'autres chercheurs. Nous avons donc testé ces sous ensemble afin d'obtenir nos résultats. Les deux sous ensemble comportent les features suivantes :

subset1 : 3,5,6,39

subset2 : 3,5,23,24,33,34,35,36

source: S. Zargari and D. Voorhis, "Feature Selection in the Corrected KDD-dataset", Third International Conference on Emerging Intelligent Data and Web Technologies, pp. 174-180, 2012, 2012.

En observant les labels présents dans chaque cluster, ainsi qu'en calculant les entropies sur les deux subset, il semble que ceux-ci produisent des résultats moins bons que le dataset complet. Bien que cela ne prouve pas que le dataset entier donnera toujours les meilleurs résultats, nous avons quand même pu voir qu'avec peu de features, les résultats sont bien moins bons (entropie environ 10 fois supérieure). Cela paraît assez évident étant donné que le clustering en règle générale apprécie posséder un nombre élevé de features. Les subsets décrits dans le document sur lequel nous nous sommes basés ont été générés lors de classification, ce peut également expliquer pourquoi ils ne fonctionnent pas avec du clustering.

Est-ce que l'algorithme K-mean est l'algorithme le plus adapté à notre problème ? Est-ce que l'algorithme K-medoids donne de meilleurs résultats ?

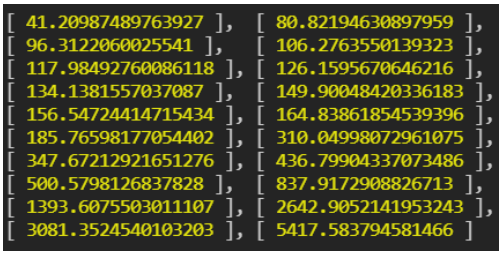
Pour cette question nous cherchions à savoir si un autre algorithme que celui des K-Means était plus adapté pour notre problème, en l'occurrence l'algorithme de PAM pour Partitioning Around Medoids. Nous pouvons directement répondre par la négative pour une simple raison, ce dernier ayant une complexité tendant vers du $O(k(n-k)^2)$, très bien expliqué dans ce document[2]. Une nouvelle approche inspirée de K-Means utilisant les Medoids[3] a été proposée en 2009 mais n'a pas été implémentée dans ce projet. Une telle complexité rend le temps d'exécution beaucoup trop long pour un dataset de presque cinq millions de points. Cependant l'algorithme est parallélisable et donc tout à fait adapté à une exécution sur Spark.

Nous avons donc souhaité tester l'algorithme sur notre dataset. Nous en avons tiré un échantillon de 10'000 points sur lesquels nous avons appliqué les algorithmes de K-Means et K-Medoids. Afin de comparer les résultats nous avons, pour chaque médoïde, fait correspondre le centroïde (K-Means) le plus proche sans en réutiliser.

Les résultats ne sont pas fondamentalement concluants. A première vue ils devraient théoriquement être meilleurs :

- PAM est beaucoup moins influencé par les valeurs atypiques que K-Means, le fait de choisir un médoïde parmi les points existants lui permet de ne pas trop décaler
- Une autre conséquence indirecte est de plus facilement éviter les minimums locaux car les sauts d'un médoïde à un autre évite de passer par un entre-deux

1. **Obliger les médoïdes à trouver le centroïde le plus proche sans réutilisation**
   Dans cette première évaluation, on observe, notamment avec k valant 100, que certains médoïdes se confondent avec certains centroïdes (distance de 0). Cependant, quelle que soit la valeur de k, on observe une dégradation rapide des distances jusqu'à avoir des points qui n'aient plus rien à voir les uns avec les autres.
2. **Avec réutilisation des centroids**
   Avec réutilisation d'un même centroïde pour plusieurs médoïdes, les centres de clusters générés par les deux algorithmes sont plus proches et dans un ordre de grandeur plus acceptable. Néanmoins ce résultat n'est pas satisfaisant car cela signifie juste réduire k en ne prenant que les centres qui nous arrangent.



Cette image illustre la distance (mean square error, ici pour k=20) en prenant pour chaque médoïde, le centroïde le plus proche sans réutilisation. Indépendamment de k nous avons observé que les deux algorithmes ne trouvent pas les mêmes centres de clusters, la distance augmentant progressivement.

[2] https://www.scitepress.org/papers/2017/65150/65150.pdf
[3] https://www.researchgate.net/publication/249891613_A_K-means-like_Algorithm_for_K-medoids_Clustering_and_Its_Performance

Est-il possible d'identifier les différentes attaques au moyen d'un algorithme non supervisé ou seule une distinction "normal/anormal" est possible ?

Pour cette question, nous cherchions à savoir si les résultats du clustering nous permettraient de déterminer le type d'attaque ou non (si un point appartient à tel cluster, alors il s'agit de telle attaque). Durant les différentes étapes de l'analyse, nous avons affiché les clusters ainsi que les labels qui les composent. Cela nous permet de répondre en partie à cette question. Globalement, la plupart des clusters sont composés d'une majorité de connexions du même type, puis de quelques connexions d'un autre type (qui sont probablement en réalité des anomalies, trop éloignés, qui changent alors de cluster).

On peut donc dire qu'il paraît possible de déterminer le type le plus probable d'attaque en fonction du cluster. Par contre, un même type de connexions peut être représenté par plusieurs clusters. C'est le cas car les connexions normales sont forcément très hétérogènes (il n'existe pas un type de connexion pour tous les protocoles), et les attaques peuvent être effectuées de plusieurs manières (par exemple, on peut faire une attaque DDOS avec peu de machines et de gros payload, ou alors avec beaucoup de machines et de petit payload). Cela crée donc de grandes variations dans les valeurs des features, et donc plusieurs clusters distincts pour chaque type de connexions.

Pour pouvoir déterminer le type d'une nouvelle connexion, il faudra donc être capable de "labelliser" les clusters découverts lors du training, en sachant que des erreurs sont possibles (notamment dans le cas d'anomalies).

Concernant les anomalies (point très éloigné de son centroïde), nous avons voulu savoir si on pouvait en savoir plus, en regardant au niveau des clusters voisins. Pour certaines attaques, il a en effet été constaté qu'un cluster voisin du bon type est présent. Donc, si un point est très éloigné du centroïde, on peut estimer qu'il peut être à la fois du type du cluster, ou alors du type d'un cluster voisin. Pour les connexions normales, il n'a pas été possible de déterminer le type d'attaque que cela pouvait représenter (tous les voisins étaient des clusters de type normal). Nous nous attendions typiquement à retrouver l'attaque à partir d'un voisin (par exemple si une connexion normale effectue des requêtes sur un grand nombre de port, nous nous attendions à observer un cluster de type port scanning proche) mais ce ne fut pas le cas

Partie 4 : Algorithme appliqué

Durant les analyses, l'algorithme K-Means a été utilisé. C'est un algorithme de type clustering, qui travaille donc en général sans label. Dans notre cas, nous possédions les labels, ce qui nous a permis de vérifier la qualité de nos résultats.
Au fil des analyses, des améliorations dans les paramètres ainsi que le pré-processing ont été implémentés afin d'améliorer les résultats. On peut par exemple citer la normalisation des features, ou encore l'augmentation du nombre d'itération maximum afin de converger plus précisément.
Dans un cas idéal, il aurait fallu pouvoir exécuter K-means un grand nombre de fois puis prendre la moyenne des résultats afin de réduire les disparités entre chaque exécution. Malheureusement, la durée d'exécution nécessaire sur notre dataset était trop longue pour pouvoir envisager cela. Les différences entre plusieurs exécutions sont dues à l'initialisation des centroïdes, qui est aléatoire. Cela conduit donc a des différences lors des convergences (il peut y avoir des minimums locaux par exemple).

Le deuxième algorithme testé a été le K-Medoids. Son fonctionnement est décrit plus en détail dans la réponse à l'une des questions d'analyse le concernant.

Partie 5 : Optimisations

Afin d'exploiter chaque feature nous avons appliqué la technique du one hot encoding afin de faire correspondre les valeurs textuelles du dataset à des valeurs numériques, comme le type de protocole (TCP/UDP). Ces valeurs sont générées initialement en parcourant le dataset afin de connaître les valeurs possibles et de créer une table de hachage de correspondance.

Partie 6 : Testing et évaluation du dataset

Dans le cas du clustering, il n'est pas nécessaire de procéder à une phase de training puis de testing. Durant le projet, nous avons commencé par chercher le bon nombre de clusters à utiliser grâce tout d'abord à l'erreur quadratique moyenne, puis au moyen de l'entropie totale. L'erreur quadratique moyenne se calcule en moyennant le carré des distances de chaque point a son centroïde. Pour l'entropie, c'est une fonction permettant de mettre en évidence les anomalies des clusters (un label n'apparaissant qu'une fois dans un cluster aura tendance à augmenter beaucoup l'entropie). Cela permet donc de détecter et d'évaluer la quantité d'erreurs de classification dans les clusters.

Une fois le bon K trouvé, nous avons ensuite cherché à analyser les résultats du clustering, en termes de quantité de labels différents par clusters, taille des clusters, ou nombre d'anomalies par clusters. Le but étant d'atteindre des résultats avec un seul label par groupe (et donc aucune erreur de clustering). Malheureusement, il n'a pas été possible d'atteindre cette qualité de résultats, notamment à cause des anomalies que nous avons détectées.

Partie 7 : Résultats obtenus

Lors de l'analyse, il a été déterminé que le nombre de clusters optimal est d'environ **180**. Jusqu'à cette valeur, les mesures d'erreur diminuent très rapidement, puis l'erreur à tendance à augmenter légèrement entre 180 et 210 clusters, avant de recommencer à diminuer, mais beaucoup plus lentement cette fois-ci, signe probable d'un début d'overfitting.
Nous avons ensuite pu dresser une liste d'anomalies dans le dataset, qui représente les points les plus éloignés de leur centroïdes (avec K-means, il n'est pas possible pour un point d'être en dehors d'un cluster, donc ce sont bien les points les plus loin qui représentent des anomalies). A partir de cette liste, nous avons essayé de déterminer plus chose :

- Dans le cas d'une anomalie de connexion normale, peut-on déterminer s'il s'agit d'une attaque ?
- Dans le cas d'une anomalie d'attaque, que peut-on observer d'intéressant ?

Pour répondre à ces deux interrogations, nous avons tenté d'observer les clusters voisins de ces anomalies. La logique voudrait qu'une anomalie lors d'une connexion normale pourrait s'approcher d'un cluster correspondant à une attaque. Cela permettrait la détection d'attaques dont la signature n'est pas encore connue. Cependant les résultats de nos analyses n'ont pas permis de confirmer cette intuition.
En ce qui concerne les anomalies dans les attaques, il semble dans ce cas que les clusters voisins correspondent bien à l'anomalie. Cette information pourrait être utile afin d'améliorer les résultats du clustering (dans le cas ou des points sont éloignés d'un centroïde, on pourrait les classer en se basant également sur les clusters voisins).

Partie 8 : Améliorations futures

Une analyse formelle des features pour déterminer lesquelles sont utiles et lesquelles ne le sont pas serait un plus pour ce projet. En effet, éliminer les informations inutiles permettrait d'accélérer le traitement et de faciliter l'analyse après coup.
Comme vu dans le notebook, une telle analyse a déjà été effectuée sur ce dataset dans le cadre de la classification et a donné deux différents subsets de features. Malheureusement, l'expérience a prouvé que ces subsets ne correspondent pas aux besoins de notre algorithme pour le clustering puisque les résultats obtenus sont bien moins bons qu'avec toutes les features. Il est donc nécessaire d'explorer différents subsets au moyen de méthode de classification (une étude à la main est, comme évoqué dans le notebook, beaucoup trop longue pour être réalisée) applicables dans le cadre d'un K-Means.
Afin de pousser également l'analyse plus loin, il semble impératif d'utiliser un cluster de machines plus performant, afin de pouvoir effectuer plus de tests, et plus rapidement.

Conclusion :

Ce projet nous a permis d'utiliser les différents outils vus pendant le cours, durant une analyse d'un cas concret. Venant d'une formation en sécurité, le sujet était d'autant plus intéressant, et permet de voir ce que peut permettre l'application du machine learning dans des problèmes réels.
Nous n'avons cependant pas réussi à développer notre analyse autant que souhaité, notamment à cause de nombreux problèmes rencontrés lors de l'utilisation des différents outils (problème de version sur zeppelin et spark par exemple, ou durée des sessions aws educate posant parfois des problèmes).