R&D challenge

I. Data science paper analysis

1) Resume, what I remember from the paper

According to the paper, while many studies are dedicated to model overall internet traffic on a webserver, few are designed to model bot traffic exclusively, which is the goal of the paper I am describing here. This would be useful to implement a realistic web traffic simulator that includes bot traffic.

In order to identify the traffic coming from bots, the authors focus on **sessions**, which consist of a sequence of HTTP requests during a specific timeframe, 30 minutes in that case. Hence, a new session is considered when 30+ minutes elapsed between two requests from a given client, and a client is uniquely identified by its IP and its user agent, both contained in every HTTP request log in the file of the webserver.
The authors focus on 3 session's features :

- session interarrival time : the time elapsed between the beginning of two sessions of a given client
- request interarrival time : the time elapsed between two requests from a given client inside a session
- number of requests : the total number of requests inside a session from a given client

The goal is to propose a model of those three features that would be close to the real traffic. To do so, they use the data of a Web server hosting a middle-sized online bookstore (osCommerce.com), a PHP-based electronic commerce platform.

This dataset is filtered to only keep the traffic coming from bots, using 7 detection rules :

- Presence of a request to the 'robots.txt' file
- A bot name in the user agent (Googlebot, AdsBot-Google, BingBot…).
- The mean time spent per page is below 0.5 seconds
- The referrer is empty for the first request of the session, all the page requests or even all the HTTP requests of the session
- All requests are HEAD instead of the more common GET
- The image to page ratio requests is zero
- Only one request in the session

. Then, using window functions, the data is transformed to isolate the sessions and compute the three features from above for each session of each day.

Finally, the authors generate non linear regression models based on the histograms obtained previously, for each of the three features. That means a different function is used to model each session feature, and by combining the three functions, it should be possible to generate a realistic bot traffic simulation.

As a result of the regression analysis, three distributional models were best fitted to the corresponding empirical data:

- a Weibull model for the session interarrival time
- a Pareto model for the request interarrival time
- a model based on sigmoid and exponential type functions for the number of requests in session

Table 3: Model of a Web Bots' Arrival Process

Traffic feature	Distribution	Parameters
Session interarrival time [s]	Weibull (3)	$A = 3{,}176.4$ $\lambda = 43.43$ $k = 0.86$ $\mu = 0$
Request interarrival time [s]	Pareto (4)	$A = 1228.6$ $k = 2.50$ $\mu = 1.43$
Number of requests in session	Sig-exp (5)	$k = -0.0003$ $b = 64801.55$ $c = 4.32$ $d = 27$ $A = 4988.86$ $B = 46.1$ $\mu_1 = -2.45$ $\mu_2 = 2.9$ $\mu_3 = 1.48$

2) Limitations

- The data used for the analysis comes from only one website, a middle-sized online bookstore. This only website might not have been targeted by all types of bots, so maybe it would have been better to use more diverse data.

- I didn't understand their method for selecting the models parameters : they choose the parameters of the model that best suited a specific day (27th april). Maybe they should have chosen an average of the parameters over all days

- I also don't understand how they validate the model. From what I understand, they implemented the model in a C++ simulation program, and they compare the generated output with the empirical data for the same day. I think this procedure might not be very rigorous : in the end, they fit a model on some training data, then they compare the output of the model with the training data, which will obviously match as the model is fitted on that same data, but maybe I misunderstood something here.

Cyril Lay
cyril31180@gmail.com

3) Interest for your company

- The paper could be implemented to create simulations of bot traffic, and test your company products on the simulation before going to production, to ensure the bots are identified and correctly classified

- Those simulations could be integrated with human-traffic simulations in order to stress test your inference pipeline with a distribution-wise realistic simulated traffic

- Data obtained from the simulations could be fed into machine learning algorithms

- Some detection rules are described in the paper, they could be evaluated by your company and eventually added to the detection engine if they aren't already, and if they happen to be relevant:
    - Presence of a request to the 'robots.txt' file
    - A bot name exist in the user agent (Googlebot, AdsBot-Google, BingBot…). Lists of known bot user-agents exist online, here and here for example)
    - The mean time spent per page is below 0.5 seconds
    - The referrer is empty for the first request of the session, all the page requests or even all the HTTP requests of the session
    - All requests are HEAD instead of the more common GET
    - The image to page ratio requests is zero
    - Only one request in the session (A user session would have multiple web requests triggered from the initial one, for images and other content)

- The statistics exposed in the paper could be useful to compare with your own statistics :

Table 1: Basic Statistics for the Server Log Data
(Timespan: 1st – 30th April 2014)

	All sessions	Bot sessions
Number of sessions	51,121	42,782 (83.7%)
Number of requests	3,200,228	1,148,863 (35.9%)
MB transferred	25,123.4	10,782.5 (42.9%)

Table 2: Statistics for Bot Sessions' Features

Statistics	**Session interarrival time [s]**	**Request interarrival time [s]**	**Number of requests in session**
Minimum	0	0	1
Maximum	6,178	1,799	47,624
Mean	61.0	28.2	26.9
Median	36	0	2
Std. dev.	83.3	142.0	242.4

If we could gather similar statistics for human sessions, both could be used to create new detection rules.

Cyril Lay
cyril31180@gmail.com

4) Improvements

- A similar analysis could be performed with server logs from multiple websites (preferably of different types : e-commerce, mobile app, entertainment website, social network, etc) instead of using only *osCommerce.com*, in order to create a better generalized model

- Instead of only using one month of data, one full year (or more) could be used to check if the trends are similar from a month to another, and if the distributions vary between months.

- Further analysis could be performed to ensure the bot detection rules' thresholds are correctly chosen (check if false positives and false negatives are present) :
    - Ensure the mean time per page threshold of 0.5 seconds is right or if it could be increased
    - Add a threshold for the HEAD/GET requests proportion, instead of only classifying the session as a bot if everything is HEAD

- Explore other features (number of sessions per day/week/month, number of HEAD/POST/GET requests, ratios between different types of requests, etc.)

- Explore other approaches for detecting bot traffic, detailed in the next part of the document

5) Other interesting approaches

- Explore client-side information : where the click happened, how did the mouse move prior to a click, etc
- Explore other rules for detecting bot traffic : lack of javascript triggers, and other rules detailed in this article.
- Explore real-time per-request classification : might give a performance challenge
- Explore using offline models generating classification thresholds (clustering, forecasting)
- Investigate using a graph based machine learning approach (https://arxiv.org/pdf/1902.08538.pdf)
- Experiment using offline clustering models generating a client blacklist

Cyril Lay
cyril31180@gmail.com

6) Detail one of them : offline clustering models

In this part, I will attempt to explain an approach that could be experimented in order to classify bot traffic without any defined rules, using unsupervised learning with the k-means clustering algorithm, and dimension reduction using PCA and t-SNE.

I didn't finish experimenting this approach, but some steps are already implemented, in the `clustering_exploration.ipynl` notebook

The steps I would follow would be :

- Generate many session features such as the number of HEAD, POST requests per session, the proportion of requests with no referrer, etc. Various session sizes can be experimented (30 minutes like in the paper, hour, day).

- Visualize plots of every combination of two features, to eventually see some patterns or clusters. Those plots are available in the appendix of this document.

- Eventually, depending on the number of features, similiar visualizations could be experimented but in 3 dimensions.

- Experiment k-means with all the features, try tweaking the K-value, and other hyperparameters such as the distance used (euclidian distance might not work so well with many dimensions)

- Experiment k-means with a reduced number of features obtained with 3 methods :
    - PCA
    - t-SNE
    - t-SNE after PCA, if the number of features is above 50[1].

- Investigate trying to initialize the k-means with a subset of sessions we know are coming from bots.

- Iterate on the whole process, spend time on feature generation

[1] It is highly recommended to use another dimensionality reduction method to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples. [source]

II. Spécification conceptuelle d'un système de détection de bots

1) Données d'entrée

En données d'entrée, nous avons un fichier `access.log`, historique complet de toutes les requêtes HTTP vers le serveur de http://www.almhuette-raith.at. Il est important de noter que ce fichier est assez gros (presque 1go), car il semblerait qu'il contient toutes les requêtes depuis décembre 2015. Dans le futur, ou dans le cas d'un autre site plus fréquenté, cela pourrait poser des soucis de scaling, car la taille du fichier ne va cesser d'augmenter, de l'ordre de 1 Mo par 10 000 requêtes[2].

Pour gérer ce problème, plusieurs solutions sont possibles :

- Selon les cas, il peut être possible de faire l'hypothèse que le journal de logs augmentera à une vitesse lente, stable (par exemple un site interne, non public, accessible seulement via un intranet, uniquement pour certaines équipes de l'entreprise). Dans ce cas, on pourrait imaginer travailler seulement à partir du fichier de log contenant l'entièreté de l'historique.

- Si l'on peut modifier le code côté serveur, il peut être intéressant d'ajouter une fonction de rotation de logs, afin d'avoir les logs partitionnés selon le jour[3]. Si l'on traite avec un client externe, cette solution peut ne pas être envisageable.

- Une autre solution serait d'avoir un job journalier ou mensuel (selon les besoins), qui se chargerait de partitionner ce fichier brut à posteriori. Plusieurs technologies peuvent être utilisées à ce niveau : un simple script Python, voir du calcul distribué, par exemple avec Spark, pour des sites dépassant le million de requêtes journalières.

Ici, pour simplifier la suite de l'étude, nous suivrons une 4ème approche : lire le fichier ligne par ligne, en commençant de la fin vers le début, en s'arrêtant à partir d'une certaine date.
Ainsi, appeler la fonction `parse_log_file(path, from_date=dt.date(2020, 3, 20))` retournera tous les logs depuis le 03 mars 2020.

[2] Source : https://httpd.apache.org/docs/2.4/fr/logs.html, partie Rotation des Journaux
[3] Une solution est proposée dans la partie "Journaux redirigés" de https://httpd.apache.org/docs/2.4/fr/logs.html

Cyril Lay
cyril31180@gmail.com

2) Conception haut niveau

Pour la conception d'un moteur de détection de bots, on peut imaginer une structure en 3 parties :

- Une partie "temps réel" qui filtre/redirige les requêtes selon des critères détectables au niveau d'une seule requête (user agent de bot, pas de referrer, IP provenant d'un datacenter...)

- Deux parties asynchrones :
    - Une pipeline sur une fenêtre 'courte' : aggréger les logs toutes les 3h, utiliser des règles simples basées sur le nombre de requêtes, le nombre de clicks, et les autres règles du papier, pour générer une liste d'IPs à blacklister : **c'est la pipeline développée dans le code du projet**
    - Une pipeline sur des fenêtres 'longues' : tous les mois, aggréger et analyser les données en semaines, et en mois, utiliser des algorithmes de clustering pour détecter les bots, utiliser les features et seuils générés par ces mêmes algorithmes pour éventuellement faire évoluer les règles de la pipeline précédente

Chaque partie pourra alimenter/puiser dans une blacklist externe, stockée dans une base de données par exemple. Cette blacklist pourrait contenir différents "niveaux" : au niveau 1, une IP est redirigée vers une page avec un captcha. Si l'IP ne résout pas le captcha, elle pourra être déplacée vers le niveau 2, avec une durée de blacklist supérieure, etc.

Cette blacklist pourrait être initialisée via un service tiers pour récupérer une base de données d'IPs connues pour être indésirables, par exemple https://www.neutrinoapi.com.

Les deux pipelines asynchrones pourraient être automatisées via différents outils ou services (AWS, Airflow, Kubernetes…), selon les outils déjà utilisés chez vous.

3) Résultats et prochaines étapes

En lançant le code de `main.py`, nous obtenons une liste d'ips qui sont potentiellement des bots. Il est à noter que cette liste est loin d'être exacte, compte tenu de la simplicité des règles de détection implémentées ici. Des analyses plus poussées sur chaque règle seraient nécessaires, afin de bien choisir les différents seuils pour s'assurer de l'absence de faux positifs.

Voici quelques statistiques sur les résultats obtenus, sur l'ensemble des données de 2015 à 2020 :

- Parmi les 68K IPs uniques, 10% des ips ont été classifiées comme bots potentiels.
- Parmi les ~5 millions de requêtes, 80% ont été classifiées comme bots potentiels. Selon cet article, environ 38% du trafic global d'internet provient de bots. Notre système de détection contient très probablement une énorme part de faux positifs.
- Parmi les ~4 millions de requêtes classifiées comme provenant de bots,
    - 18% contenaient une requête au fichier `robots.txt`
    - 3% contenaient les mots "bot", "crawler" ou "spider" dans leur user-agent
    - 58% ont effectué, au moins une fois, plus de 100 requêtes sans referrer en une heure, avec un temps moyen entre requêtes inférieur a 20 secondes lors de cette heure
    - 54% ont effectué, au moins une fois, plus de 1000 requêtes sans referrer en une journée, avec un temps moyen entre requêtes inférieur à 20 secondes lors de ce jour
    - 91% ont effectué, au moins une fois, plus de 100 requêtes en une heure
    - 96% ont effectué, au moins une fois, plus de 1000 requêtes en une journée

Ces deux derniers seuils sont probablement ceux qui ont été le moins judicieusement choisis. En effet, en m'aidant de la console chrome, j'ai pu remarquer que déja 33 requetes étaient envoyées au serveur en chargeant la page d'accueil : http://www.almhuette-raith.at/. En rafraîchissant 4 fois la page en moins d'une heure, un humain serait déjà classifié comme un bot.

Afin de poursuivre ce travail, de nombreuses pistes pourraient être explorées. En premier lieu, il faudrait passer plus de temps sur les règles ci-dessus, les évaluer plus en détail pour les adapter. Ensuite, de nombreuses autres features pourraient être recherchées et évaluées. Un autre papier[4] utilise par exemple les features suivantes :

Table 1. Data set features and their meaning

Feature	Meaning
F1	Total number of sent control TCP packets per connections
F2	Total number of received TCP control packets
F3	Total number of TCP control packets
F4	Total length of sent TCP control packets
F5	Total length of received TCP control packets
F6	Average length of sent TCP control packets
F7	Average length of received TCP control packets
F8	Average length of TCP control packets

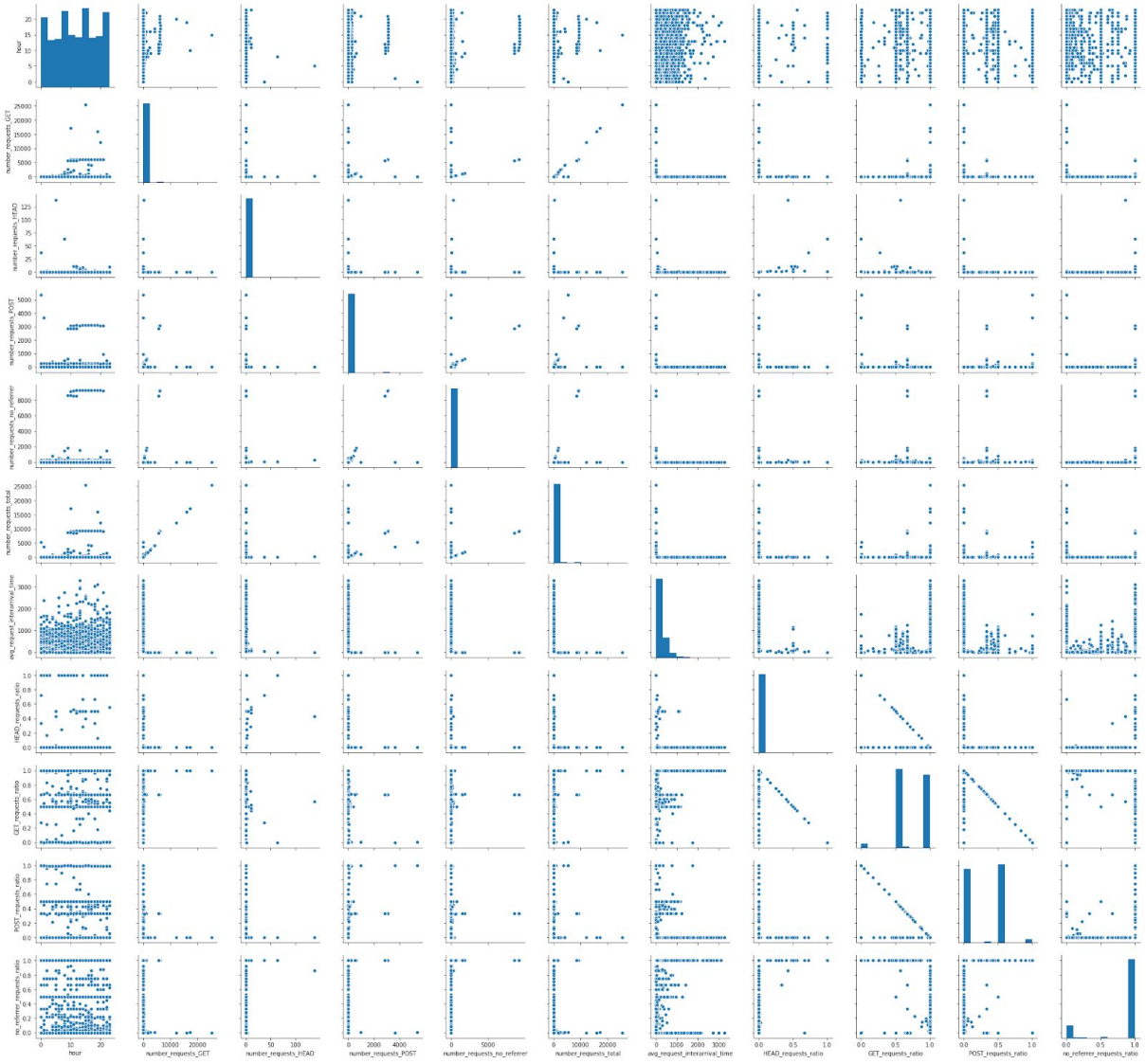
[4]https://www.researchgate.net/publication/318325614_Hybrid_Approach_for_Botnet_Detection_Using_K-Means_and_K-Medoids_with_Hopfield_Neural_Network

Enfin, la partie clustering est à compléter. Une prochaine étape serait d'intégrer la composante de temps dans les features, par exemple avec des colonnes contenant les valeurs des features de la session précédente.

Cyril Lay
cyril31180@gmail.com

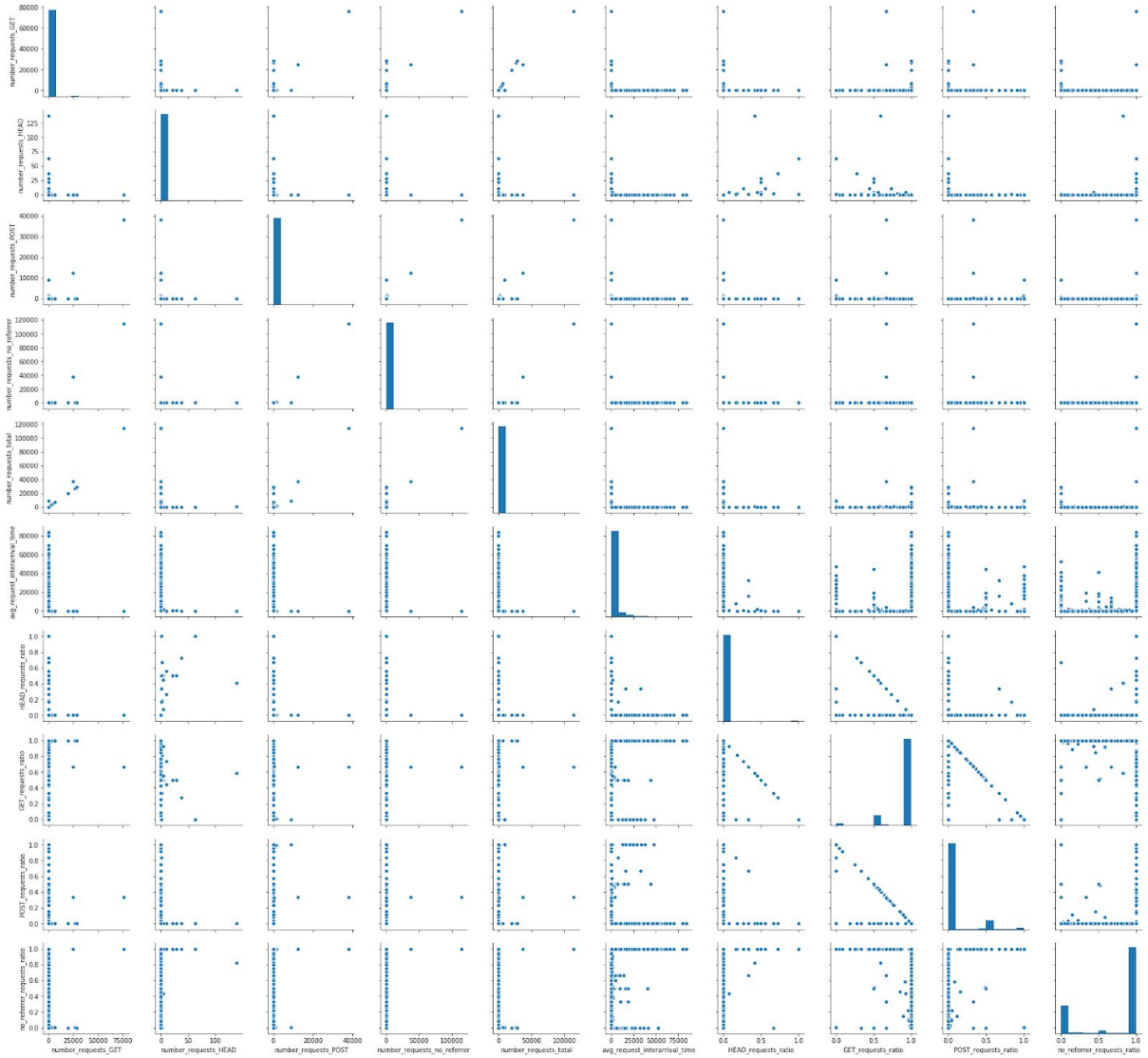
Appendix

1. Hour sessions features pair-wise plots



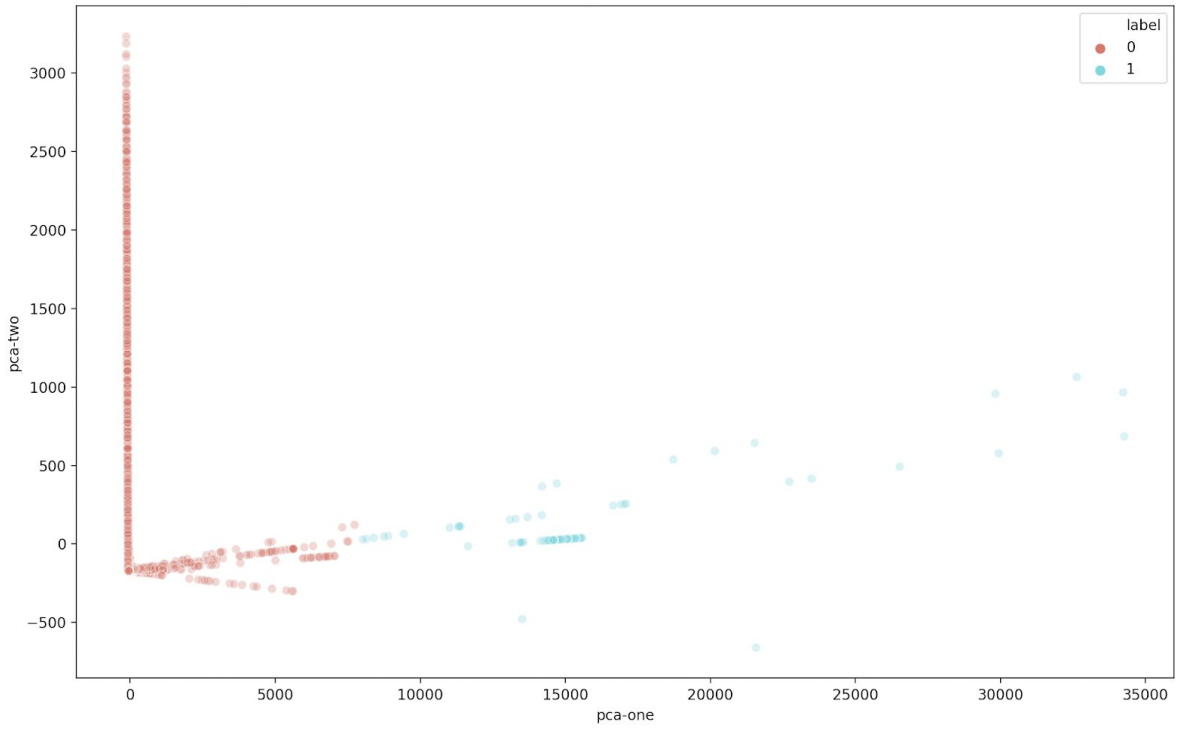
Cyril Lay
cyril31180@gmail.com

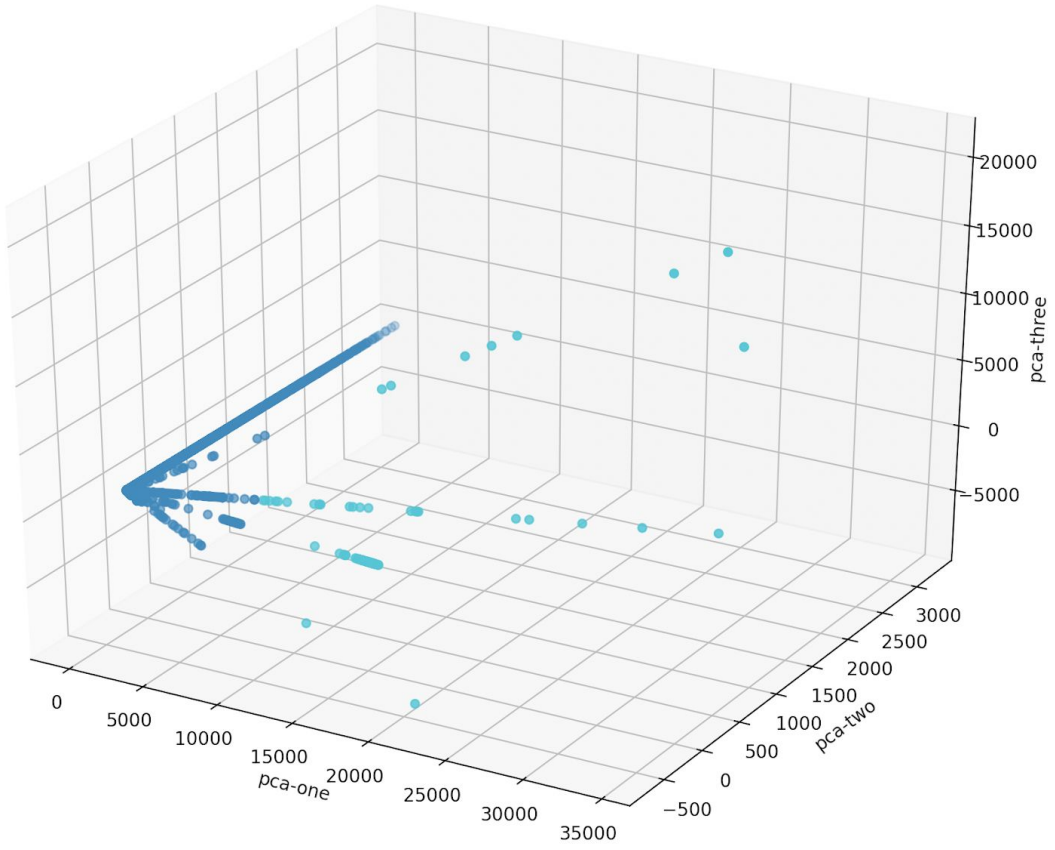
2. Day sessions features pair-wise plots



Cyril Lay
cyril31180@gmail.com

3. K-means result on PCA-reduced data





Cyril Lay
cyril31180@gmail.com