Ecole Publique d'Ingénieurs en 3 ans

Rapport

IMPLÉMENTATION DE FILTRES D'ANALYSE EN INVESTIGATION NUMÉRIQUE

Le 30 mars 2023,

Abosede ALAKINDE, Gatien BOUYER, Basil COURBAYRE, Téo DALLIER, Félix DAUNE
Étudiants

Clients/Tuteurs : **Christophe ROSENBERGER, Emmanuel GIGUET**

www.ensicaen.fr

TABLE DES MATIÈRES

Contenu

TABLE DES FIGURES

REMERCIEMENTS

Avant toute chose, nous souhaitons remercier les diverses personnes sans qui ce projet n'aurait pas été possible.

Tout d'abord, nous adressons nos remerciements à nos tuteurs et clients M. Christophe ROSENBERGER, et M. Emmanuel GIGUET, chercheurs au GREYC, qui nous ont accompagnés en tant que tuteurs et clients toute l'année. Ils ont été réactifs à l'ensemble de nos demandes et se sont montrés compréhensifs lorsque nous rencontrions des difficultés. Ils nous ont beaucoup aidés à comprendre le cadre du projet et la plateforme G'DIP.

Nous tenons à également remercier les développeurs front-end et back-end de G'DIP que nous connaissons à travers leurs pseudonymes : Hugo Jean, Adrien et H. Merly. Ils nous ont beaucoup accompagnés quand nous avions besoin d'aide pour nos filtres, sur la recherche de solutions ou de data sets. Leur aide nous a permis d'avancer plus aisément dans le projet.

PRÉSENTATION DU CONTEXTE

1. Présentation générale du projet

Notre projet intitulé « Implémentation de filtres d'analyse en investigation numérique » se focalise sur le milieu de la forensique[1]. En informatique, c'est un ensemble de méthodes pour récupérer les données détruites et les métadonnées. Elle désigne également les méthodes pour traiter toutes ces données et extraire les informations pertinentes. Dans la plupart des cas, la forensique est utilisée pour les affaires criminelles ou judiciaires.

Dans le cadre de notre projet, notre équipe, composée de Abosede ALAKINDE, Gatien BOUYER, Basil COURBAYRE, Téo DALLIER, et Félix DAUNE, a été amenée à travailler avec nos tuteurs et clients Christophe ROSENBERGER et Emmanuel GIGUET. Ils sont à l'initiative de la plateforme G'DIP.

2. Présentation de G'DIP

Afin de comprendre les aboutissants de notre projet, nous devons d'abord présenter la plateforme logicielle G'DIP (GREYC Digital Investigation Platform). Cette plateforme, qui est en cours de développement, permettra, à terme, à des investigateurs comme des policiers ou des chercheurs d'analyser des masses de données pour obtenir des informations contenues dans ces données. Pour ce faire, la plateforme est composée de nombreux filtres (recherche de texte, catégorisation de fichiers, analyse d'images, etc.).

Plus précisément, un filtre est un outil qui permet d'analyser des données pour en extraire des informations. Par exemple, si l'on trouve une clé USB dont on ignore le propriétaire, on pourrait se servir de divers filtres pour retrouver le propriétaire et lui rendre son bien. En effet, grâce à G'DIP, il sera, à l'avenir, possible de récupérer l'ensemble des fichiers présent dans la mémoire quel qu'en soit le format, et même si le fichier a été supprimé. On pourra ensuite appliquer des filtres pour trouver mails, noms, numéros de téléphone, etc. parmi l'ensemble des fichiers. On pourra également analyser d'éventuelles images pour voir quels sont les centres d'intérêts représentés (villes, paysages, pornographie, …) ou connaître la personne la plus photographiée. Mais à ce jour, dans la situation donnée, tous ces filtres n'existent pas encore. C'est ici que notre projet entre en jeu.

3. Présentation du besoin

Comme expliqué précédemment, G'DIP est en cours de développement. Ce qui signifie que l'application a un besoin constant de nouveaux filtres pour pouvoir extraire le plus

[1] La forensique regroupe les méthodes d'analyse scientifique pour l'investigation.

d'informations possible à partir de données brutes. Après discussion avec nos clients, nous avons convenu que les filtres les plus intéressants et les plus urgents devaient extraire les informations des fichiers textes et catégoriser les visages dans les fichiers image, c'est donc ce sur quoi notre équipe projet s'est focalisée.

Plus particulièrement, le premier filtre que nous avons développé, s'intéresse aux adresses mails, c'est-à-dire que ce filtre devrait permettre de reconnaître, dans un fichier quelconque, toutes les adresses mails quel qu'en soit le format.

Le second filtre s'intéresse quant à lui aux numéros de téléphone. De la même manière que le précédent, celui-ci devrait reconnaître tout type de numéro de téléphone dans tout type de fichier.

Enfin, le dernier filtre traite des images, et plus particulièrement, il s'intéresse au domaine de la reconnaissance et la classification de visages. Le but de ce filtre est de pouvoir regrouper par personne, diverses photos contenues dans un espace de stockage. Sur le long terme, cela permettrait de savoir quelles sont les différentes personnes présentes dans les images fournies et lesquelles sont les plus fréquentes.

Ainsi, en cumulant ces trois filtres, le problème de la clé USB serait plus facilement solvable, car grâce à eux, on pourra savoir quels numéros de téléphone et mails la personne a dans ses documents et on pourra éventuellement connaître la tête de la personne. Ces filtres, cumulés à d'autres (existants ou à venir), permettraient de dresser un véritable portrait-robot de la personne. Et dans un contexte plus large, ils permettraient de connaître l'entourage et les fréquentations de cette personne.

OBJECTIFS ET MÉTHODOLOGIE

4. Présentation des objectifs du projet

Les premiers objectifs avaient pour objectif de comprendre le projet, son contexte et sa faisabilité.

L'objectif initial était l'étude de l'existant car nous ne devions pas réinventer la roue. Il était préférable d'aller trouver des solutions existantes et de les adapter pour créer nos filtres.

Ensuite, un second objectif était de choisir les data-sets. En effet, chaque filtre a besoin de data-sets pour être testé. Plus les data-sets sont complets et hétérogènes, plus le filtre aura de chance d'être complet et efficace sur les cas pratiques rencontrés sur le terrain.

À partir de cette étude de l'existant, nous devions ensuite choisir une ou plusieurs solutions, et les implémenter dans des Proof Of Concept[2] (POC) que nous devions présenter à nos clients. Ces POC devaient nous permettre d'évaluer la faisabilité des filtres.

L'objectif final de notre projet était de créer 2 à 4 filtres. Cependant, après discussion avec nos clients, 3 filtres parfaitement fonctionnels seraient amplement satisfaisants. Après discussion, les filtres choisis sont ceux évoqués précédemment : le filtre *mail*, le filtre *numéro de téléphone* et le filtre *classification de visages*.

Pour permettre leurs intégrations à G'DIP, ces filtres se devaient d'être documentés, intégrables et rendus avec un manuel d'utilisation (et/ou documentation).

5. Méthodologie utilisée

5.1. Répartition du travail

Afin de mener ce projet à bien, nous avons scindé notre équipe projet en 2 sous-groupes.

Abosede et Félix ont respectivement été attribués à la recherche de mails et de numéros de téléphone dans des fichiers textes.

Quant à Gatien, Basil et Téo, ils ont travaillé sur la création d'un trombinoscope des visages les plus présents dans une banque d'image.

Par ailleurs, pour mener ce projet à bien, nous avons choisi Félix comme chef de projet.

[2] Méthode qui permet d'évaluer la faisabilité d'un projet.

5.2. Méthode de travail

Pour ce qui est de la méthode de travail utilisée, nous avons décidé de travailler avec une méthode pseudo-agile, car nous jugions que c'était une méthode adaptée au projet. En effet, nous avions prévu de travailler de manière incrémentale avec des retours fréquents avec nos clients.

Pour travailler de manière agile, nous avons scindé notre équipe en petits groupes, ce qui a facilité le travail par binôme/trinôme. Nous avons également fait des POC, qui est typique de la méthode agile et nous les avons amélioré de manière incrémentale.

Par ailleurs, nous avons fait de l'intégration continue sur notre dépôt Git. Nous avons également réalisé des tests fréquents pour vérifier l'efficacité et la performance des filtres. Enfin, nous avons suivi un autre principe clé de l'agilité en faisant des retours réguliers avec nos clients, idéalement toutes les 2 à 3 semaines.

5.3. Outils utilisés

Pour ce qui est des outils utilisés, nous nous sommes servis de Discord pour communiquer avec les clients dans un serveur créé par eux et dédié à G'DIP. Nous avions également un serveur pour notre équipe pour communiquer plus aisément entre nous.

Nous avons utilisé également Git pour gérer les versions de code.

Enfin, nous avons utilisé des éditeurs de code comme Visual Studio Code et PyCharm pour réaliser nos codes et tests.

Figure 2 : Logo Git

Figure 1 : Logo Discord

6. Éléments de solutions aux besoins

Avant de se concentrer sur les différents filtres, il y a une exigence qu'ils ont tous en commun. En effet, les filtres se doivent d'être intégrables dans G'DIP. Ils doivent donc être codés dans le langage de programme Python, mais avec la possibilité d'utiliser des librairies Python tierces, ce qui était fortement encouragé par nos clients.

6.1. Filtres sur fichiers texte

6.1.1. Adresses mail

Concernant le filtre d'adresses mails, il était clair qu'il allait falloir trouver un moyen de détecter tout type d'adresse mail. Mais c'est bien ici que l'obstacle principal du filtre se dressait. Il fallait être capable de détecter les adresses mails quel que soit leur format. En effet, si l'on trouve la majorité des adresses mails dans le format *`prenom.nom@domaine.extension`*, il peut toutefois arriver que les adresses soient modifiées pour résister aux scrapeurs[3] qui essayent de récupérer des adresses mails sur les sites. C'est pour cela que l'on peut trouver des adresses mails sous des formats différents tels que *`prenom.nom[at]domaine(dot)extension`*. Ainsi, nous savions à l'avance que notre solution se devait de s'adapter à tous ces types d'adresses mail.

Finalement, le filtre de mails avait trois grandes fonctionnalités à remplir :

- Il devait pouvoir reconnaître tout type d'adresses mails, quel que soit leur format.
- Naturellement, il fallait également que le filtre ait un taux de reconnaissance très élevé et un taux d'erreur le plus faible possible.
- Le filtre devait également venir avec un petit manuel d'utilisation pour aider à sa compréhension et faciliter son intégration à G'DIP.

6.1.2. Numéros de téléphone

Pour ce qui est du filtre de reconnaissances de numéros de téléphone, les objectifs étaient très similaires au précédent. En effet, les numéros de téléphones ont eux aussi beaucoup de formats différents. Les numéros peuvent être au format national ou international, les chiffres peuvent être séparés par des points, des tirets, des espaces ou d'autres séparateurs encore.

Ainsi, on se retrouve avec les mêmes exigences que le filtre précédent mis à part que les formats à reconnaître seront différents.

6.2. Filtre de classification de visages

Pour ce qui est du filtre de classification de visages, nous savions que les données de départ étaient des photos non labellisées[4] de personnes. Ensuite, à partir de ces photos, il fallait pouvoir retrouver les visages les plus fréquents. C'était donc la contrainte principale de ce filtre : classifier des images en fonction des visages pour savoir quelle personne est la plus présente est quelles sont ses images associées.

[3] Outil allant sur des sites pour récupérer des informations comme des adresses mails
[4] C'est à dire sans avoir de connaissance sur l'identité de la personne sur l'image

De la même manière que les filtres précédents, ce filtre devait également répondre aux attentes de G'DIP. Ce filtre pouvant être plus complexe que les précédents, le manuel d'utilisation était d'autant plus souhaitable. Le filtre devait également être relativement efficace en termes de rapidité et avoir un taux de classification satisfaisant.

PRÉSENTATION DES SOLUTIONS TROUVÉES

7. Méthodologie de résolution des problèmes

7.1. Filtres sur fichiers texte

Comme évoqué précédemment, la première étape pour ces filtres était la recherche de data sets. Cependant, pour éviter de nous compliquer la tâche et pour y aller petit à petit, nous avons décidé de créer des petits data sets à la main pour tester plus facilement. Cela ne serait qu'après-avoir eu des résultats concluants avec ces premières données que nous aurions testé nos filtres sur des data sets disponibles sur internet. L'idée était de créer des fichiers suffisamment petits et couvrant un maximum de situation pour pouvoir tester les filtres rapidement. C'est pour cela que l'on a créé trois fichiers de contenu identique, mais avec des extensions différentes : un *.pdf*, un *.odt* et un *.txt*. Ces fichiers étaient constitués d'adresses mails ou de numéros de téléphone sous différents formats et mêlés avec du texte quelconque. Vous trouverez en Figure 3 un exemple de fichier utilisé pour les adresses mails.

Fichier de test d'adresses mails. jean@google.com, george @ outlook. fr Lorem opsum dolir sit amet.
Le paul (at) gmail (dot) com fin de phrase

jean.rene (at) orange.com
a.b@z.com

test.1	@	dansun.tableau
test.2	at	dans (dot) untableau

Figure 3 : Exemple de data set initial pour les mails

Dans cette figure, on peut voir que les adresses mails sont sous divers formats. On a des adresses mails classiques dans des phrases, des adresses mails séparés par des espaces, des adresses mails non conventionnelles avec les (at) et (dot) et des adresses mails dans des tableaux.

Pour ce qui est des numéros de téléphone, nous avons également construit un data set qui est représenté en Figure 4.

Ici encore, on peut voir des formats différents comme les numéros de téléphone au format classique, international et avec différents types de séparateurs.

Une fois cette première étape réalisée, nous nous sommes penchés sur le problème des fichiers. En effet, comme nous l'avons dit, les filtres devaient être capable de traiter tout type de fichier. Nous avons donc dû rechercher une librairie pour permettre ceci. Heureusement,

nos tuteurs nous ont recommandé *Tika* une librairie python qui répond entièrement à nos besoins. En effet, cette librairie permet de détecter et extraire du texte et des métadonnées depuis des fichiers textes de plus de 1000 types différents (*ppt*, *xls*, *pdf*, …). Cette étape de recherche fut donc relativement brève et nous n'avions plus qu'à tester la librairie, ce qui fut également assez rapide car la documentation fournie était relativement complète.

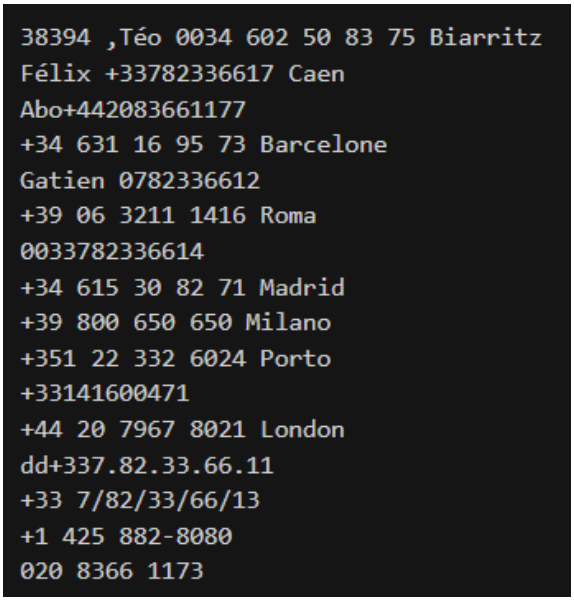

Figure 4 : Exemple de data set initial pour les numéros de téléphone

Ensuite, nous avons dû comprendre les résultats fournis par Tika. En effet, afin de pouvoir les exploiter pour reconnaître les adresses mails et les numéros de téléphone, nous devions avant tout savoir sous quel format Tika nous fournissait les données. Par exemple, lorsque la librairie lit des tableaux, elle les formate en ajoutant un retour à la ligne et une tabulation pour chaque case de chaque ligne, et effectue un saut de ligne entre deux lignes du tableau. Il fallait donc les prendre en compte.

Puis, il fallait passer à la phase de reconnaissance des adresses mails et numéros de téléphone. C'est ici que le travail de Félix et Abosede s'est divisé en 2.

7.1.1. Adresses mail

En ce qui concerne les mails, il fallait pouvoir naviguer des données textuelles rapidement et retrouver les données facilement. Heureusement pour nous, c'est exactement le besoin auquel répondent les expressions régulières[5] (nommées regex). Ainsi, le travail, et principal verrou de ce filtre, était de trouver une regex unique permettant de détecter toutes les adresses mails dans les données textes. Pour réussir ceci, nous avons procédé de manière incrémentale en commençant avec une expression régulière classique, et en l'améliorant petit à petit pour qu'elle fonctionne avec tout type de mails. La première regex est visible en Figure 5. Elle était

[5] Chaine de caractères permettant de reconnaître un motif dans une chaîne de caractères

relativement simple, mais elle permettait de reconnaître les adresses mail les plus classiques, et mêmes des plus complexes avec par exemple la reconnaissance de masquages de type (at) ou [dot].

```
([a-z0-9_.-]+ ?\(?\[? ?(@|at) ?\)?\]? ?(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])? ?\(?\[?
?(\.|dot) ?\)?\]? ?)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)
```

Figure 5 : Première itération de la regex de reconnaissance d'adresses mail

Une fois cette regex créée, il a ensuite fallu l'améliorer pour qu'elle soit complète. Comme mentionné, c'était le verrou principal de ce filtre car il fallait trouver des nouveaux data sets à chaque fois pour essayer de rendre la regex la meilleure possible.

C'est pour cela que, pendant les améliorations de regex, nous avons également recherché de meilleurs data sets pour pouvoir avoir la regex la plus complète possible. Pour trouver ce data set, nous nous sommes servis (entre autres) de GitHub, où, grâce à l'aide de nos tuteurs, nous avons trouvé un jeu de données très complet (Challenge_TextMine_2023[6]) proposant des mails sous toutes leurs formes et avec deux data sets, un composé de données factices et un avec des données réalistes.

Pour ce qui est du cycle d'amélioration des regex, nous avons d'abord géré le cas des tableaux en *.odt*. En effet, avec *Tika*, les cases du tableau étaient traduites par des sauts de ligne, suivis par des tabulations. Il a donc fallu gérer ce cas dans la regex.

Ensuite, nous avons également constaté la présence de données inutiles dans les résultats. Cela était dû aux groupes de capture des regex qui retournaient les données dans des tuples. Ainsi, ces groupes de capture ont été supprimés pour contourner ce problème.

Une étape suivante était de reconnaître plus de mails que la regex actuelle. Cette dernière était intéressante mais trop limitée (prise en compte de caractères spéciaux, formats différents, …). C'est pour cela que nous avons entamé des recherches pour trouver de meilleures regex. La meilleure et plus complète trouvée vient de emailregex.com (le site semble ne plus être disponible). Nous avons donc rassemblé la regex originale avec celle trouvée en ligne.

[6] https://github.com/Emvista/Challenge_TextMine_2023

Enfin, la dernière amélioration a été de permettre à la regex de reconnaître les lettres majuscules, et certains caractères spéciaux comme à, é, è et ê. Ces dernières modifications ont permis d'approcher une regex sans erreur.

7.1.2. Numéros de téléphone

En ce qui concerne les numéros de téléphones, notre objectif était d'avoir un filtre qui détecte les numéros de téléphone dans un fichier texte même s'ils étaient cachés. Nous avons commencé une première itération de prototype avec des regex. Il fallait trouver une regex capable d'extraire tout type de numéros de téléphone. Nous avons trouvé la regex suivante.

```
'/\d{10}|\+33\d{9}|\+33\s\d{1}\s\d{2}\s\d{2}\s\d{2}\s\d{2}|\d{2}\s\d{2}\s\d{2}
\s\d{2}\s\d{2}|0\d(?:\.\d{2}){4}|0\d(?:-\d{2}){4}/'
```

Cependant, cette regex posait divers problèmes :

- Elle ne reconnaissait que les numéros commençant par le numéro indicatif de la France du type : +33738475647
- Elle ne prenait pas en compte les séparateurs dans les numéros de téléphone.
- Elle ne trouvait pas les numéros cachés dans un texte.

Ainsi, pour résoudre ces problèmes, nous avons modifié notre prototype pour utiliser une librairie Python : *phonenumbers*[7]. Cette librairie avait pour avantage de permettre la reconnaissance des numéros du monde entier, pas seulement les numéros français. De plus, elle permet de reconnaitre les numéros de téléphone utilisant des séparateurs tels que : « *espace*, **.** , **-** , **()** ». Par exemple, un numéro du type : +33 (0)4.27.13.38.34 est reconnu par le filtre. Cette librairie a donc permis de résoudre un bon nombre de problèmes posés par la regex.

Cette librairie n'était néanmoins pas parfaite car elle ne prenait pas tous les séparateurs en compte. Parmi eux, les séparateurs : « **/**, **:** , **;**, **,** ». On peut cependant noter que ces séparateurs sont très peu utilisés. Nous avons donc fait le choix de conserver cette librairie malgré ce léger défaut. Nous en avons cherché d'autres, mais n'en avons pas trouvé d'aussi complète que *phonenumbers*.

Le verrou principal de ce filtre était de trouver des data sets complets pour effectuer les tests. En effet, nous cherchions des fichiers contenant des numéros de téléphones, des longs chiffres, du texte et d'autres données si possible, mais nous n'avons pas réussi à en obtenir aisément. La meilleure solution que nous avons trouvée a été de créer ces data sets à la main. Par la suite, nos clients nous ont fourni des data sets avec des numéros de téléphone, nous avons pu faire des tests sur ces données.

[7] https://pypi.org/project/phonenumbers/

7.2. Filtre de classification de visages

Pour ce qui concerne l'équipe chargée de la classification des visages, le problème était multiple. Le filtre se décompose en trois parties plus ou moins centrales : la détection des visages sur les images, la caractérisation d'un visage pour pouvoir ensuite le comparer avec d'autres visages et enfin le regroupement des visages par même personne.

Avec recommandation de nos tuteurs, nous avions pris le data set LFW comme base de test pour ce filtre. Dans ce dernier, il y a 13 233 photos, soit 5 749 personnes (avec une ou plusieurs images par personne) mais sur ces personnes, ne sont gardées que les images où un seul visage est détecté. Dans le cas échéant, il y aurait confusion entre les visages pour savoir lequel est celui du label (c'est-à-dire la personne associée à cette photo). À la suite du tri, il ne resta alors que 11 201 photos, soit 5 137 personnes avec entre 1 et 430 photos par personne.

Une fois le data set établi, il a fallu trouver une manière de détecter les visages sur les images. Pour résoudre ce premier verrou, l'équipe s'est tournée vers internet pour chercher des solutions existantes. À la suite des recommandations de nos tuteurs, la librairie MTCNN a été sélectionnée. Cet outil se base sur les réseaux de neurones pour détecter les visages. Le modèle de détection de visages derrière MTCNN est le plus utilisé dans la littérature scientifique, d'où notre choix.

Puis, une seconde étape était de calculer les caractéristiques des visages. En effet, afin de pouvoir différencier et classifier les visages, nous devions les caractériser. Là aussi, nos clients nous avaient indiqué une solution avec *keras_vggface*, une librairie python pour le calcul de caractéristiques de visages.

Enfin, il a fallu trouver une dernière librairie pour classifier les images. Or, pour rappel, nous traitions des images non labellisées, nous étions donc dans un cas de classification non supervisé et il fallait trouver des algorithmes de clustering[8]. Ceci constituait le point clé de notre filtre, mais aussi son verrou principal, car il y avait beaucoup de possibilités, et nous n'avions pas beaucoup d'expérience dans le milieu.

Au départ, nous avons commis la maladresse d'écrire un algorithme à la main pour faire le clustering alors que l'idée est déjà implémentée sous le nom plus large de classification hiérarchique (voir plus bas le paragraphe sur *scikit-learn*).

Le principe est de réunir itérativement les groupes de caractéristiques de visage similaires jusqu'à ce que tous les groupes soient trop distants. Initialement, chaque visage est dans son propre groupe, puis, on réunit les groupes deux par deux selon un critère (la distance entre deux caractéristiques par rapport à un seuil). Si la distance entre deux éléments de deux groupes différents, appelée la distance minimale entre les deux groupes, est inférieure au seuil

[8] C'est à dire des algorithmes qui créent des groupes de données sans avoir d'informations sur ces données au préalable.

alors ils sont réunis en un seul groupe. Il y a ici, 3 paramètres à choisir : le seuil, la distance et le critère de réunion (outre la distance minimale, il y a la distance maximale, la distance moyenne et d'autre sur la variance).

Ainsi, nous avons tenté d'implémenter ce clustering hiérarchique par nous-même. Nous avons choisi de tester des seuils différents avec une distance cosinus comme conseillé dans la référence[9] fourni par nos clients. Les performances de l'algorithme atteignent au mieux 90% de bonnes associations avec le seuil à 0.29 sur l'ensemble du data set LFW (cf. Figure 6).

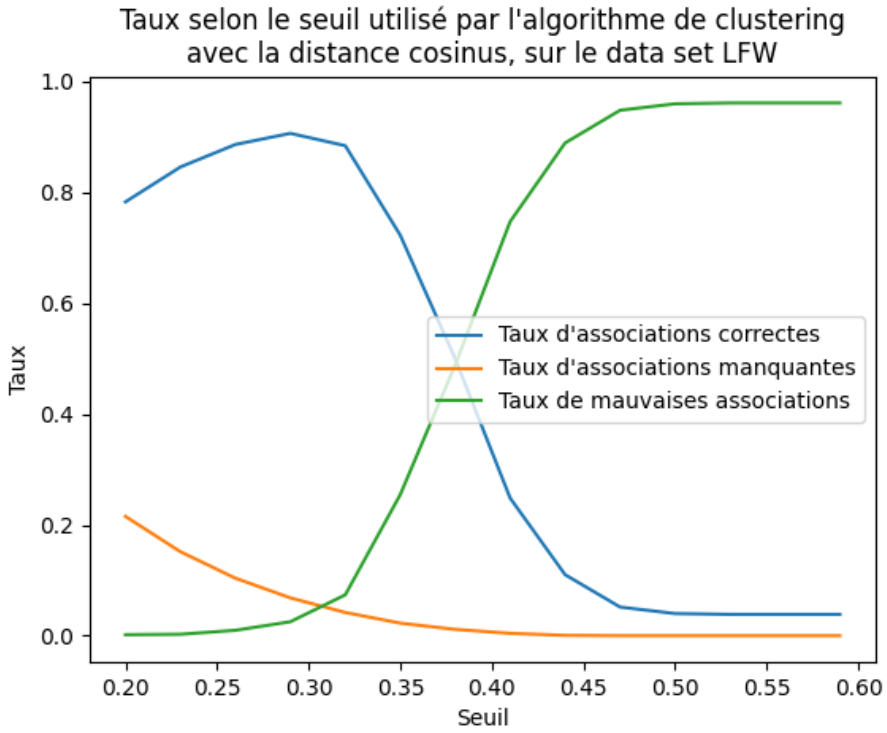

Figure 6 : Graphique des taux de d'association par type selon le seuil utilisé

Pour évaluer les prédictions produites par l'algorithme de regroupement, nous avions construit un algorithme pour associer les prédictions avec les labels attendus fournis pour le data set. Les groupes de visages prédits doivent représenter chacun une et une seule personne. Les erreurs de prédictions sont alors de 2 types. Le premier arrive lorsque plusieurs personnes différentes se retrouvent prédites dans le même groupe et sont ainsi traités comme étant une seule et même personne. Les associations entre les visages de toutes ces personnes sont donc mauvaises et non voulus. Le second, à l'opposé, arrive quand une même personne se retrouve séparée dans divers groupes et ainsi n'est pas bien reconnue car les associations entre les visages de cette personne manquent. Une illustration de la méthode de la catégorisation des associations est disponible **Error! Reference source not found.**. Nous pouvions ainsi

[9] Référence principale utilisée pour les choix de technologie pour le filtre image : https://machinelearningmastery.com/how-to-perform-face-recognition-with-vggface2-convolutional-neural-network-in-keras/

déterminer la part d'associations correctes, la part de mauvaises associations et la part d'associations manquantes.

Finalement, après avoir trouvé une solution convenable, nous avons décidé de nous rabattre vers une solution existante bien plus rapide. La librairie *scikit-learn*[10] de python semblait parfaite pour remplacer notre algorithme, car elle propose plusieurs algorithmes de clustering tels que la classification hiérarchique comme celle que nous avions utilisée jusque-là, mais aussi d'autres algorithmes de clustering comme DBSCAN. De plus, la librairie propose également des méthodes pour évaluer le résultat du clustering. Nous avons par exemple utilisé l'information mutuelle. Par manque de temps nous n'avons pu qu'évaluer l'*AgglomerativeClustering* de *scikit* qui est l'implémentation directe de notre algorithme décrit plus haut. Ce dernier produit les mêmes résultats que le nôtre, mais s'exécute bien plus vite.

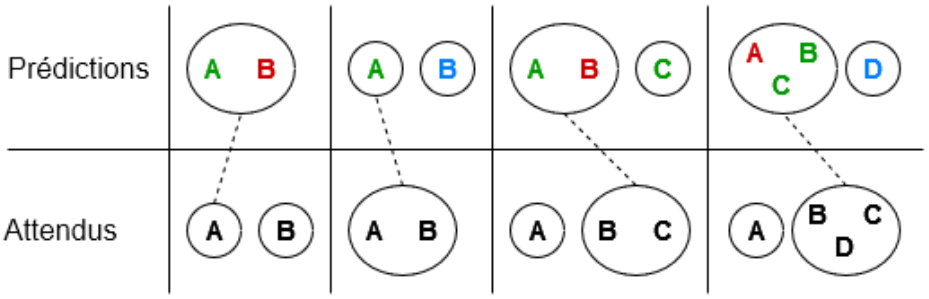

Figure 7 : Méthode de catégorisation des associations des prédictions avec les résultats attendus. En vert, les visages correctement associés, en rouge ceux mal associés, et en bleu ceux non associés

En parallèle de toutes les étapes citées précédemment, nous avons également cherché des meilleurs data sets. En effet, nous avions LFW qui est relativement bien fourni, mais comme très souvent en apprentissage, il faut faire des tests avec plusieurs data sets pour diversifier les données. Nous avons donc cherché des data sets qui correspondaient à notre besoin très spécifique, ce qui s'est avéré être un problème de taille. En effet, nous voulions trouver des data sets avec des photos labellisées pour chaque visage pour pouvoir vérifier nos prédictions. De nombreux data sets répondaient en partie au problème, mais aucun n'avait tous les visages labellisés. Par exemple, il y avait un data set labellisé de groupe intéressant, mais le nom des personnes n'y figurait pas, nous avions uniquement le genre et l'âge des personnes. Nous aurions eu besoin de labels indiquant le nom de chaque personne dans l'image. Ainsi, nous avons utilisé un autre data set qui se rapprochait le plus à nos besoins : le Celebrity-Face-Recognition-Dataset[11]. D'une taille de 172 Gb, il contient près de 800 000 images de 1100 célébrités. Ce data set a été très utile pour perfectionner notre filtre en gérant des erreurs que nous n'avions pas eues avant comme les images en noir et blanc ou encore les images dans des formats différents comme des couleurs 8 bits et non 24.

[10] https://scikit-learn.org/stable/modules/clustering.html
[11] https://github.com/prateekmehta59/Celebrity-Face-Recognition-Dataset

8. Récapitulatif des solutions trouvées

Finalement, nous avons réussi à avoir trois filtres relativement fonctionnels.

Le premier filtre, permettant de reconnaître les adresses mails, a été testé de manière extensive et permet de reconnaître tout type d'adresses mails, qu'elles soient masquées ou non. Nous avons décidé d'arrêter notre recherche avec cette expression régulière (Figure 8), car elle a fonctionné sur toutes les adresses mail de nos jeux de données. Nous avons également pris en compte la complexité de l'expression régulière, car plus elle est complexe, plus la recherche d'éléments dans un texte devient longue. Bien que ce temps reste minime par rapport à d'autres types de traitement tels que le traitement d'images, il était important pour nous de maintenir des temps de recherche rapides.

```
(?:[a-zA-Z0-9éèà!#$%&'*+/=?^_`{|}~-]+(?:\.[a-zA-Z0-9éèà!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x0
1-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")
?\(?\[? ?(?:\n\t)?(?:@|at)(?:\n\t)? ?\)?\]? ?(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?
?\(?\[? ?(?:\.|dot) ?\)?\]? ?)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-
4][0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-
9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x
0b\x0c\x0e-\x7f])+)\])
```

Figure 8 : Regex finale pour le filtre de reconnaissance de mails

Le deuxième filtre, reconnaît les numéros de téléphone du monde entier et les classe par pays selon la demande de l'utilisateur qui peut choisir plusieurs pays. Le filtre est fonctionnel, il n'extrait que des numéros de téléphone. Des tests ont été fait sur des fichiers qui contiennent des longs chiffres, du texte ainsi que des numéros de téléphone. Seuls les numéros de téléphone valides ont été extraits. Sur ce filtre, la partie de traitement des séparateurs spéciaux reste à traiter même s'ils sont rares.

Le troisième filtre, permettant de regrouper les images par personnes, atteint des performances convenables sur nos jeux de données. Cependant, l'application du filtre à des cas pratiques montre ses limites (Figure 9). Il faudrait continuer l'étude pour trouver les paramètres optimaux pour les algorithmes de clustering et en tester d'autres. La partie reconnaissance de visage et la partie de calcul des caractéristiques des visages n'ont pas à être plus poussées que la version fonctionnelle que nous avons, car elle fera certainement partie, à terme, d'un autre filtre spécifique sur la plateforme G'DIP. Ces parties sont néanmoins très impactantes sur les données en entrée de l'algorithme de clustering.

Figure 9 : Résultat du filtre sur les photos de l'Elysée d'octobre 2022

La personne 1 semble être Emmanuel Macron, la personne 2 semble être un groupe d'erreur (faux visages, personnes différentes), la personne 3 semble être Olaf Scholz et la personne 4 semble est Charles Michel.

9. Tests

Afin de vérifier la validité de nos filtres, nous avons également fait des tests d'efficacité. Les résultats sont les suivants.

Pour le filtre sur les adresses mails, l'efficacité est totale. Nous n'avons eu aucune erreur, sur les 668 éléments des data sets (respectivement 371 et 297 adresses mails), et toutes les adresses mails sont bien reconnues.

Pour le filtre sur les numéros de téléphone, nous avons utilisé les mêmes data sets que pour la reconnaissance d'adresses mail : JDF (factice, 372 numéros) et JDR (réaliste, 474 numéros) avec lesquels nous avons pu tester l'efficacité de notre filtre. Pour le data set JDF, 57 numéros sur 474 n'ont pas été reconnus et 8 sur 372 pour JDR. Ces valeurs restent relativement faibles et sont dues à la présence de séparateurs spéciaux dans les numéros de téléphone des data sets. La précision totale est de 92%.

Pour le filtre sur le regroupement des images par personne, la classification atteint un taux de succès de 90,7% sur le data set LFW (2,5% des visages sont associés à des mauvaises personnes et 6,8% des visages ne sont pas associés avec les autres de la même personne). Le seuil optimal est de 0.29 pour la distance cosinus et la distance minimale comme critère de fusion des clusters.

BILAN

Notre projet avait un objectif en apparence très simple : réaliser des filtres utilisables par G'DIP. Mais de nombreuses contraintes accompagnent cet objectif. Si nous en avons rempli une bonne partie, nous n'avons pas réussi à aller aussi loin que voulu sur certains autres points.

10. Objectifs non atteints, motifs et conséquences

10.1. Ce qui a fonctionné au sein du groupe

Dans ce projet, nous avons fait une bonne répartition des tâches et division du groupe, car il y avait des personnes compétentes en python attribuées à chaque filtre.

Nous avions également une bonne entente dans le groupe et avec les tuteurs/clients.

De bons moyens de communication ont rapidement été établis dans l'équipe et avec les clients via des serveurs Discord (un pour l'équipe et un avec les clients), avec lesquels nous communiquions sur nos avancées et problèmes toutes les 2 semaines environ.

10.2. Ce qui n'a pas fonctionné au sein du groupe

Nous avons eu des problèmes d'organisation avec certaines tâches qui étaient mal réparties ou mal abouties. La conséquence était une perte de temps ou du travail qui n'aboutissait à rien.

Nous avons également commis l'erreur de nous focaliser sur une unique solution pour la classification des visages, alors que l'idée originale était d'en considérer plusieurs en même temps.

10.3. Partie textuelle

10.3.1. Adresses mails

Pour la partie des adresses mails, nous estimons avoir rempli les objectifs intégralement, car le filtre rempli les demandes du client. Il est fonctionnel, simple, documenté et efficace. Il est également intégrable à G'DIP.

10.3.2. Numéros de téléphone

En ce qui concerne la partie traitement des numéros de téléphones, nous avons quasiment atteint nos objectifs.

Nous aurions aimé pouvoir rajouter à notre filtre la reconnaissance des séparateurs spéciaux. Ceci s'explique par le manque de temps pour parcourir en profondeur l'entièreté de

la librairie choisie afin de trouver les parties de la librairie qui traitent des séparateurs pour ensuite pouvoir intégrer les séparateurs spéciaux.

Ainsi, nous n'avons pas pu atteindre un taux d'erreur nul pour ce filtre, mais le taux d'erreur atteint reste satisfaisant.

10.4. Partie classification de visages

Pour la partie classification, même si nous avons bien avancé sur le filtre, nous aurions aimé tester d'autres méthodes et paramètres pour savoir si le résultat était améliorable.

Par exemple, pour la reconnaissance de visages, nous utilisons un réseau de neurones qui, après recherche, semble être la meilleure méthode pour détecter des visages. Cependant, nous n'avons utilisé que la librairie python MTCNN et nous n'avons pas pu tester d'autres librairies comme face-recognition. De la même manière, pour le calcul de distance entre deux classes, nous avons utilisé le regroupement hiérarchique, car cela nous a semblé comme l'option la plus naturelle au moment de cette étape. Nous aurions voulu tester d'autres algorithmes comme DBSCAN qui aurait aussi pu être efficace, mais nous n'avons pas eu le temps de le tester. Au final, si notre filtre fonctionne, il n'est peut-être pas optimal en termes de temps et d'efficacité.

Par manque de temps, nous n'avons pas non plus eu le temps d'intégrer le filtre à G'DIP. La conséquence est qu'un développeur de G'DIP sera chargé de le faire, mais nous l'assisterons si nécessaire dans le processus en restant sur le serveur Discord de G'DIP. Nous avons également fourni un code commenté et avec un README pour faciliter sa compréhension.

11. Projection sur la suite du projet

Finalement, nous avons abouti à 3 filtres fonctionnels. Si toutefois des tests sur des data sets supplémentaires seraient toujours bienvenus, nos filtres se sont montrés capables d'atteindre leurs objectifs. En effet, nous parvenons à extraire toutes les adresses mails des documents avec le premier filtre. Le deuxième filtre extrait la majeure partie des numéros de téléphone. Et le troisième filtre permet, avec plus ou moins de précision, de réunir les images par personnes représentées. Pour reprendre notre exemple de la clé USB perdu, nos filtres donnent efficacement accès aux adresses mail et numéros de téléphone du potentiel propriétaire et de son entourage, ainsi qu'à leurs visages présumés. La suite consiste en l'intégration de nos filtres à la plateforme G'DIP. Dès lors, l'utilité effective de nos filtres sera visible et la confrontation aux cas pratiques permettra de continuer à améliorer nos filtres si nécessaire.

12. Conclusion

Pour conclure, nous sommes satisfaits de ce que ce projet a pu nous apporter. Nous avons pu mettre en application plusieurs aspects de l'informatique en utilisant des notions d'apprentissage dans le milieu de la forensique qui nous était inconnu auparavant. À nos dépens, nous avons aussi constaté qu'en informatique et plus particulièrement dans des projets agiles, bien que la planification soit utile, un diagramme de Gantt n'est pas nécessaire étant donné qu'il est vite voué à ne plus être suivi, car on se base sur les retours du client. Nous avons apprécié pouvoir suivre un projet du début à la fin, car trop souvent les projets donnés dans le cadre des cours sont trop scolaires et non applicables à de véritables applications. Enfin, nous avons apprécié pouvoir choisir et discuter avec nos clients des différentes possibilités et filtres à implémenter, cela nous a permis de nous impliquer plus facilement et plus vite dans le projet.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

ministère éducation nationale enseignement supérieur recherche

Normandie Université

Cti Commission des titres d'ingénieur

cdefi Conférence des Directeurs des Écoles Françaises d'Ingénieurs

European Accreditation of Engineering Programmes EUR-ACE®

CONFÉRENCE DES GRANDES ÉCOLES