IMT Lille Douai

Rapport de projet : Authentification faciale

UV - ODATA

Ingrid Flogny

2019-2020

Table des matières

I – Objectifs du projet

L'objectif du projet est de mettre en place et d'évaluer un système d'authentification facial automatique. Ce système consiste à déterminer si une personne peut avoir accès ou non à un service donné.

Le système comprend un ensemble de photos correspondant aux utilisateurs enregistrés. L'authentification consistera donc à vérifier si le visage de l'utilisateur demandant l'accès correspond bien au visage d'un utilisateur enregistré dans la base de référence.

II – Description succincte des systèmes mis en œuvre

1. Création des sets de données

Les données auxquelles nous avions accès étaient deux ensembles contenant des images :

- dataset1 contenant 7462 images de visages de 373 personnes différentes ;
- dataset2 contenant 6059 images de visages de 1203 personnes différents.

Pour chaque dataset, nous avons constitué 3 groupes de données aléatoirement :

- une « **gallery** », considérée comme la base de référence, contenant des photos d'utilisateurs autorisés à l'accès du système ;
- un ensemble « **data_test_match** » constitué de 100 requêtes. Ces données sont des photos d'individus ayant accès au système (d'autres images d'eux sont enregistrées dans gallery) ;
- un ensemble « **data_test_unmatch** » constitué de 100 requêtes. Ce sont des photos d'individus n'ayant pas accès au système (aucune image d'eux n'est enregistrée dans gallery).

Dans la suite, nous nommerons gallery1, data_test_match1, data_test_unmatch1 les groupes de données de dataset1. Et gallery2, data_test_match2, data_test_unmatch2 les groupes de données de dataset2.

Afin de créer ces 3 jeux de données, nous avons tout d'abord étudié la nomenclature des photos. Elles étaient enregistrées au format suivant : X.Y.png avec X l'identifiant de l'individu sur la photo et Y, l'identifiant de la photo parmi celles d'un même individu.

Pour créer les 3 jeux de données de dataset1 (le principe est le même pour dataset2), nous avons utilisé 3 fonctions : **premier_point(st**), **appartenance_individu(photo,liste,G)** et enfin **dataset_separation1()**. Voici leur principe de fonctionnement :

- **premier_point(st**) : prend en argument le nom d'une photo et renvoie l'identifiant de l'individu.
- **appartenance_individu(photo, liste, gallery)** : teste si l'individu de photo est présent dans liste (contenant des photos d'individus). Renvoie un booléen (« True » s'il y a appartenance et « False » sinon).
- **dataset_separation(path_vers_images)** : Sélectionne aléatoirement 200 photos d'individus différents parmi les photos de la galerie d'images (en utilisant la fonction appartenance_individu) et stock l'indice des photos dans h. Crée un « probe_test_match » en y ajoutant 100 photos de gallery correspondant aux indices des 100 premiers éléments de h. Remplace ces photos dans gallery par '0'.
  Pour créer « probe_test_unmatch » nous avons procédé similairement mais en remplaçant également par '0' le nom de toutes les photos dans gallery correspondant à l'individu dont la photo se trouve dans « probe_test_unmatch » pour que ces individus n'aient pas accès au système.
  Enfin il faut supprimer toutes les '0' de gallery et renvoyer gallery, probe_test_match et probe_test_unmatch.

- **test_doublon(liste)** : teste s'il y a au moins deux photos d'un même individu dans une liste de photo (renvoie « True » si c'est le cas). Ce programme a été créé pour s'assurer que les jeux de données étaient correctement construits.

On effectue ces opérations avec les dataset 1 et 2 pour avoir nos 3 autres jeux de données par dataset.

2. Constituer la vérité-terrain

La vérité-terrain nous permet d'évaluer la performance de notre modèle. Pour la constituer, nous avons considéré que le système renvoyait deux réponses possibles : Accès autorisé et Accès refusé. Les accès autorisés ont été modélisés par des « 1 » et les accès refusés par des « 0 ».

```
verite_terrain=[1 for _ in range(100)]+[0 for _ in range(100)]
```

En testant nos systèmes d'authentification, nous créerons donc un tableau avec les résultats des 200 requêtes et il sera aisé de construire la matrice de confusion entre la vérité terrain et nos prédictions.

III – Systèmes d'authentifications

1. Fonctions utiles

Pour exploiter nos données, nous avons créé un algorithme nommé **matrisation_lineaire(liste de photos, path_vers_images)** prenant en paramètre une liste de nom de photos et renvoyant une matrice de taille (n.d) avec n le nombre d'individu (donc la longueur de la liste) et d le nombre de pixel de la photo (d=22500).

Attention : la fonction **imread** du package **matplotlib.image** nous renvoie une matrice d'octets et non d'entiers (un niveau de gris étant codé avec un octet car 256 valeurs possibles). Le problème est que si l'on effectue des opérations élémentaires avec des octets, le résultat sera modulo 256.
Par exemple : octet1=145 et octet2=153 alors octet1-octet2=-8 modulo 256 = 248.
Le problème va donc être pour la distance entre deux vecteurs qui n'aura plus de sens. Pour remédier à ce problème on utilise la commande permettant de forcer le type des valeurs d'un tableau numpy :

```
sortie=sortie.astype('int')
```

Un extrait de la matrice **G_linéaire1 :**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	151	151	152	152	149	148	148	147	144	141	137	128	117	105	88	78	61	48
1	149	149	149	150	151	148	146	146	144	142	136	129	117	102	85	68	56	49
2	141	139	140	143	142	140	137	135	134	134	127	123	113	103	95	77	66	56
3	143	141	140	138	135	136	138	139	138	130	118	105	93	83	72	61	50	37
4	149	146	144	141	137	137	139	139	138	137	132	125	109	95	88	74	65	54
5	150	149	150	152	148	144	144	144	141	142	137	129	120	105	89	82	67	54
6	148	146	146	146	144	144	141	137	136	130	126	117	103	92	76	64	55	43
7	147	146	145	143	142	142	141	138	137	132	124	111	100	88	71	60	50	35
8	151	152	153	152	150	151	152	150	148	144	137	124	108	95	83	71	59	46
9	150	146	144	144	142	141	141	140	139	137	131	119	107	95	80	65	55	38
10	152	151	151	148	145	145	146	144	146	140	138	127	111	100	85	70	56	49
11	152	150	150	150	148	147	147	146	147	146	142	138	126	114	105	88	79	66
12	141	138	137	138	134	131	133	136	137	128	117	101	88	77	63	55	46	39
13	145	144	147	150	149	147	146	143	140	136	131	124	117	110	97	83	66	55
14	148	147	146	145	143	143	141	136	133	129	123	110	98	88	73	65	53	46
15	146	145	146	145	141	140	140	139	136	126	116	105	93	87	78	64	57	49
16	146	147	145	140	138	138	140	140	141	139	134	124	113	103	91	79	67	56

Figure 1 - Extrait de la matrice G_lineaire1

2. Système d'authentification par force brute

Nous avons développé un système d'authentification par force brute par recherche de voisins proches dans un rayon r. Pour cela nous avons considéré que deux images I et I' représentent une même personne si la distance qui les sépare est inférieur à un seuil r donné : $d(I, I') \leq r$, chaque image étant représentée par un vecteur appartenant à $\mathbb{R}^d$. Nous avons utilisé pour nos calculs la distance euclidienne.

Radius_search(r,nom,G) : Prend donc en paramètre un rayon, une requête et la matrice G_lineaire1 de la gallery1 (sous la forme d'une matrice (n*d)). Renvoie la liste des voisins dont la distance est inférieure à r, au sens de la distance euclidienne.

Par exemple : `Radius_search(4000, 'lcelli.18.jpg', G_lineaire1)`

Renvoie : `['lcelli.15.jpg', 'lcelli.16.jpg', 'lcelli.17.jpg', 'lcelli.19.jpg', 'lcelli.20.jpg']`

3. Système d'authentification par force brute avec Eigenfaces

Il s'agit de développer un système de recherche de voisins proches par force brute sur des données dont la dimension a été réduite à l'aide de la méthode des Eigenfaces. On cherche à appliquer l'ACP sur gallery1 et gallery2.

1 ère étape : linéariser les n images comme fait précédemment en créant G_linéaire1 et G_lineaire2

2ème étape : Centrer-réduire les données : on centre-réduit G_lineaire grâce à la méthode StandardScaler de sklearn.preprocessing.

3ème étape : Effectuer une ACP sur la matrice linéaire centrée-réduite gallery1. Nous avons calculé les valeurs propres de la matrice de covariance. Lorsque nous traçons la courbe des valeurs propres, nous obtenons ceci :

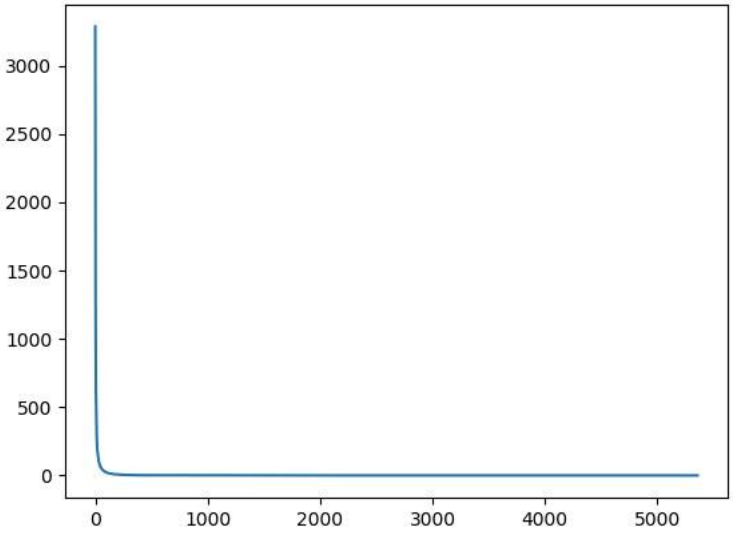


Figure 3 - Courbe des Valeurs propres de gallery1

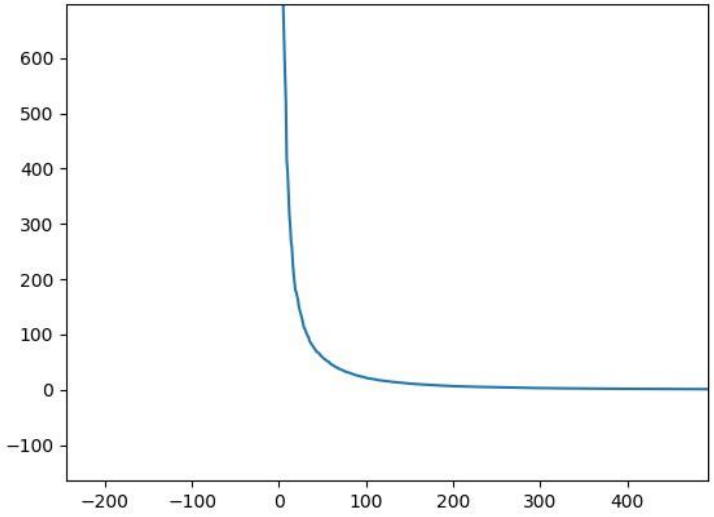
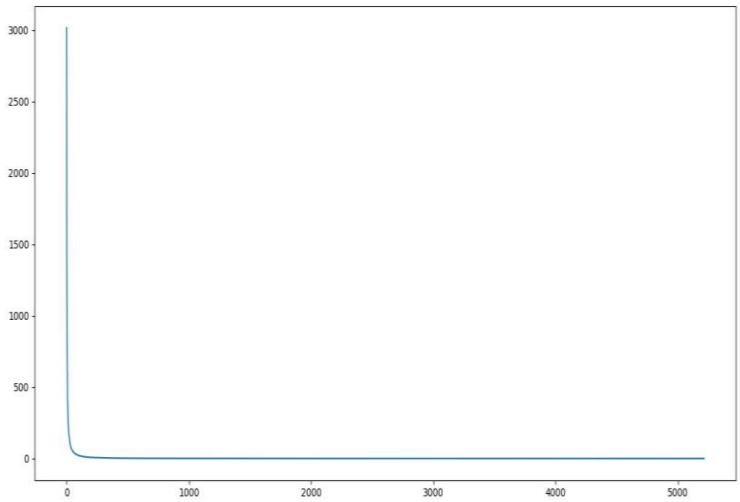


Figure 2 - Zoom sur le coude de la figure 2

On observe un coude au niveau d'une valeur propre d'environ 50, d'après la méthode de l'éboulis des valeurs propres, on conserve toutes les valeurs propres supérieures à 50. La règle de Kaiser nous indique qu'il faut prendre toutes les valeurs propres >1. En réalité, nous allons tester 4 cas différents, les valeurs propres étant supérieures à : 1, 10, 50 et 100.

Valeur propre minimale	Nombre d'axes conservés	Qualité globale (%)
1	542	98
10	163	93
50	58	82
100	33	75

On effectue le même travaille sur gallery 2 :



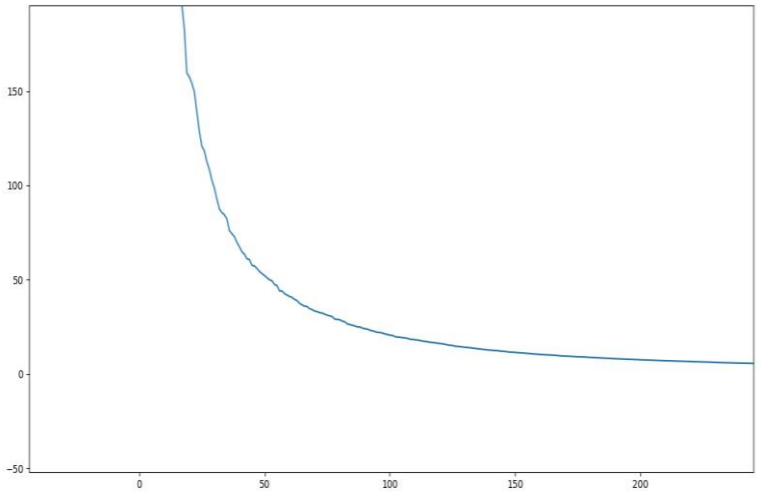


Figure 4 - Courbe des Valeurs propres de gallery2 Figure 5 - Zoom sur le coude de la la figure 4 Conclusion de l'ACP effectuée sur la gallery2 :

Valeur propre minimale	Nombre d'axes conservés	Qualité globale (%)
1	803	96
10	167	88
50	53	77
100	30	70

On voit déjà que le second jeu de données est de moins bonne qualité que le premier avec la qualité globale inferieure dans le premier jeu pour un nombre d'axes bien superieurs.

Etape 4 : Projeter les images sur les k composantes principales.

On va projeter les images des gallery dans le premier plan (déterminé par les composantes principales 1 et 2).

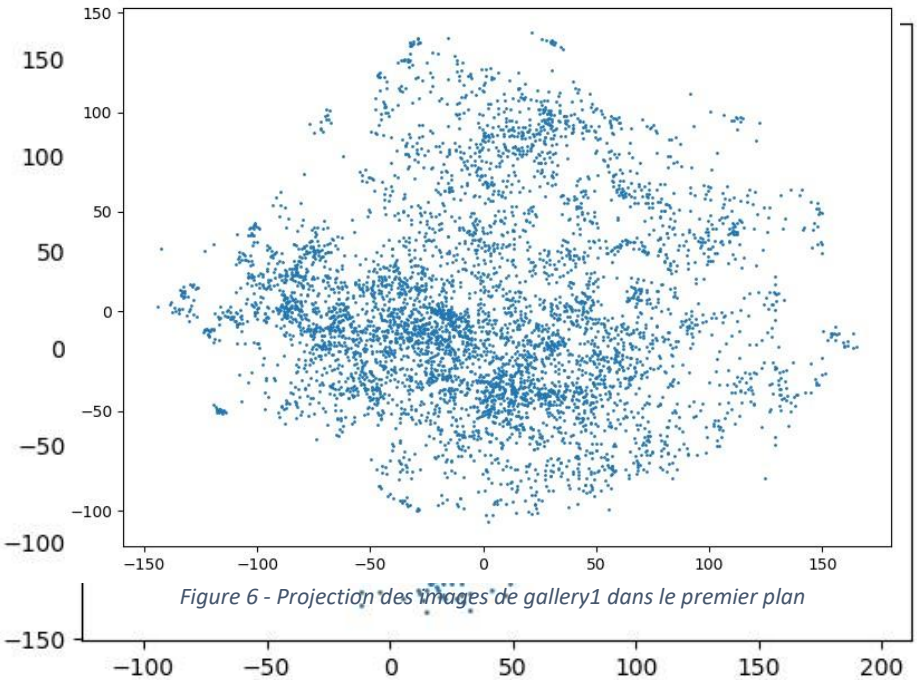


Figure 6 - Projection des images de gallery1 dans le premier plan

Figure 7 -Projection des images dans le gallery2 dans le premier plan

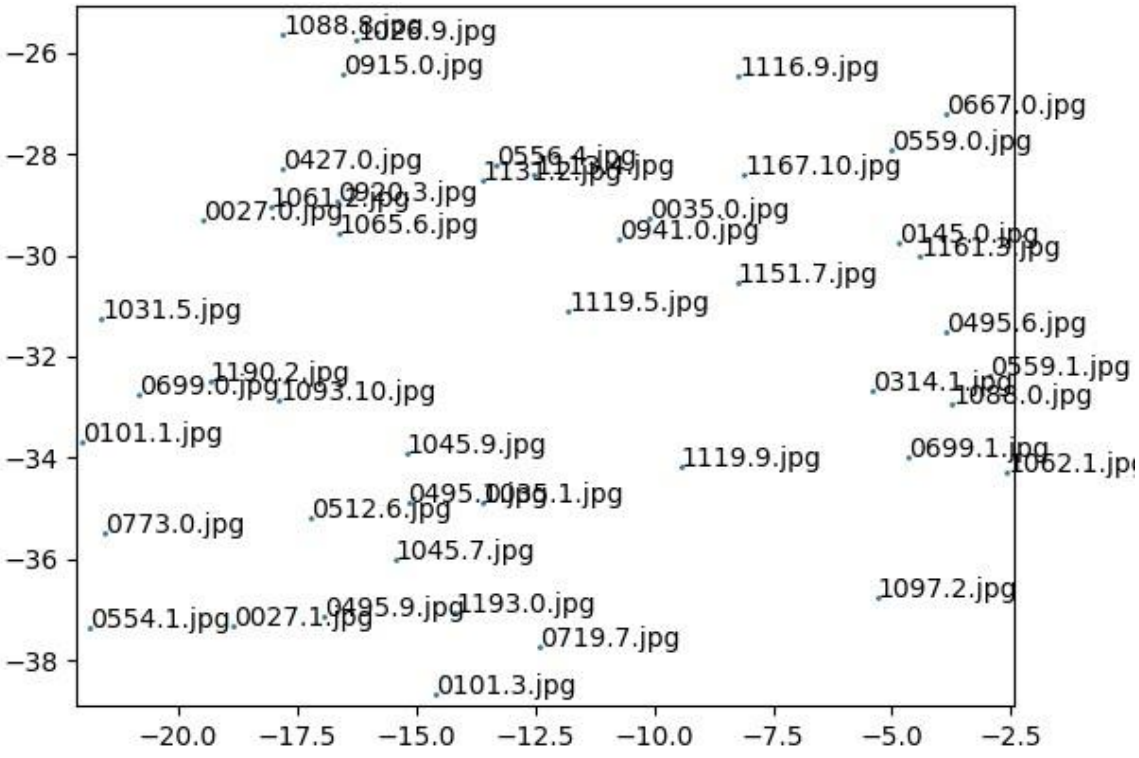


Figure 8 - Zoom sur le nuage de points de la figure 7

Etape 5 : Recherche brute (en utilisant le radius Search) avec les vecteurs projetés

Après avoir projeté les vecteurs de G_linéaire sur les k composantes principales, on obtient une matrice de dimension (n*k) qu'on appelle G_lineaire_cr.

IV. Calcul des paramètres optimaux

1. Evaluation d'un système

Pour évaluer la qualité de notre système, nous disposons de plusieurs mesures de performances :

- **Exactitude :** $A = \frac{TP+TN}{TP+FP+TN+FN}$ – Mesure le taux de bonnes réponses produites par le système ;
- **Précision :** $P = \frac{TP}{TP+FP}$ – Mesure le taux de personnes correctement autorisées parmi l'ensemble des personnes autorisées par le système ;
- **Sensibilité (ou rappel) :** $Sen = \frac{TP}{TP+FN}$ – Mesure le taux de personnes correctement autorisées parmi l'ensemble des personnes qui devraient être autorisées par le système.
- **Spécificité :** $Spe = \frac{TN}{TN+FP}$ – Mesure le taux de personnes correctement refusées parmi l'ensemble des personnes refusées par le système.

Nous avons également décidé de rajouter les deux mesures de performances suivantes :

- **F1 score :** $F1 = 2 * \frac{P*Sen}{P+Sen}$ – Mesure montrant le bon équilibre entre la précision et la sensibilité.
- **Courbe ROC** : Courbe avec les FP en abscisse et les TP en ordonnée. La situation idéale se trouve dans le coin supérieur gauche lorsque les TP sont maximaux pour des FP minimaux.

Pour déterminer les TP, FP, TN, FN on utilise la matrice de confusion implémenté dans Python sous la forme *sklearn.metrics.confusion_matrix(vérité terrain, prediction).* Cette fonction renvoie une matrice C telle que la valeur C_{ij} soit égale au nombre d'observations de i dans la vérité terrain prédit en tant que j dans la prédiction.

Notre matrice de confusion va donc être de taille (2*2), on attribue chaque coordonnée à une mesure d'évaluation :

- **(0,0) :** Correspond aux utilisateurs n'étant pas enregistrés et n'ayant pas eu d'accès (**TN**)
- **(0,1) :** Correspond aux utilisateurs n'étant pas enregistrés et ayant eu un accès (**FP**)
- **(1,0) :** Correspond aux utilisateurs enregistrés et n'ayant pas eu d'accès (**FN**)
- **(1,1) :** Correspond aux utilisateurs enregistrés et ayant eu un accès (**TP**)

Pour résumer on a ces valeurs avec la commande :

```
tn, fp, fn, tp = confusion_matrix(verite_terrain,resultats).ravel()
```

2. Force brute

Un système d'authentification basé sur le Radius Search dispose de deux paramètres à faire varier pour autoriser l'accès à un utilisateur : le rayon r et le nombre de photo minimal (i) dans la boule. Nous avons donc testé 40 combinaisons différentes pour chaque jeu de données avec des rayons valant : 2000, 3000, 4000, 5000, 6000, 7000, 8000 et 9000 et des i valant : 1, 2, 3, 4 et 5.

a) Pour le dataset1

On a donc fixé un i et tracer les mesures de performances en fonction du rayon. Pour le cas i=1 on obtient les courbes suivantes (pour les autres cas, voir en annexe) :

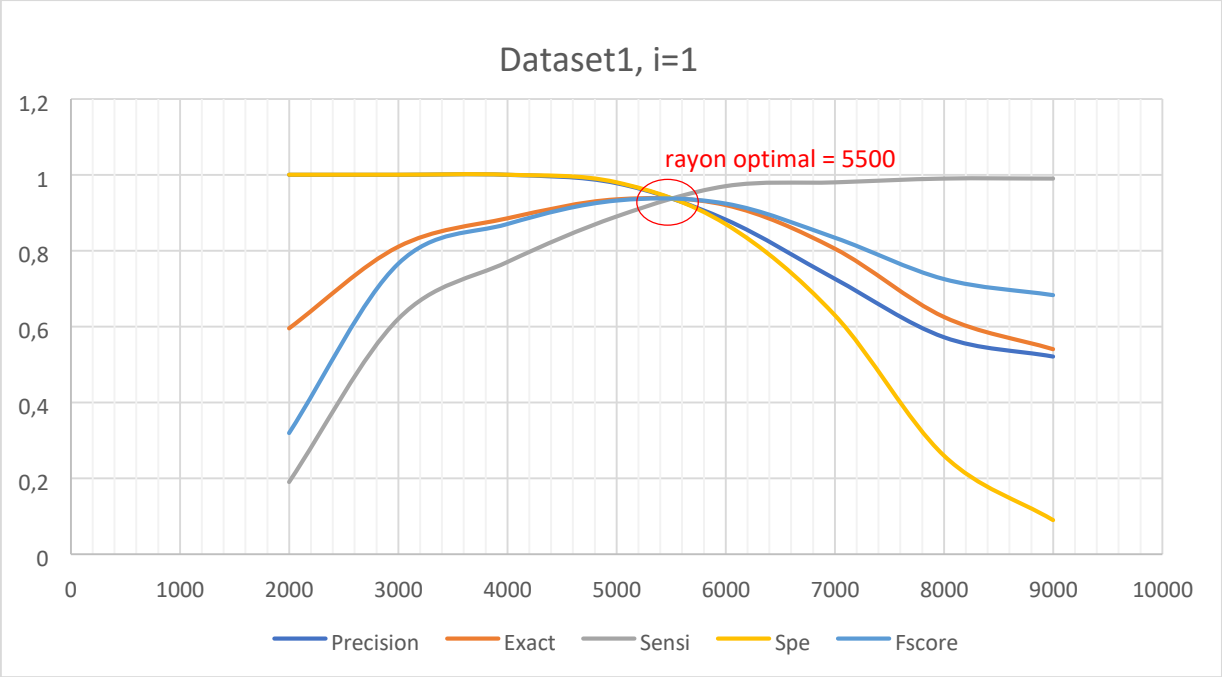


Figure 9 - Mesures de performances pour le dataset1 et i=1 en fonction de r

En traçant ces différentes mesures pour des i différents, on s'est aperçus que le rayon optimal variait selon i. Plus étonnant, il variait de manière linéaire pour les 5 premières valeurs de i :

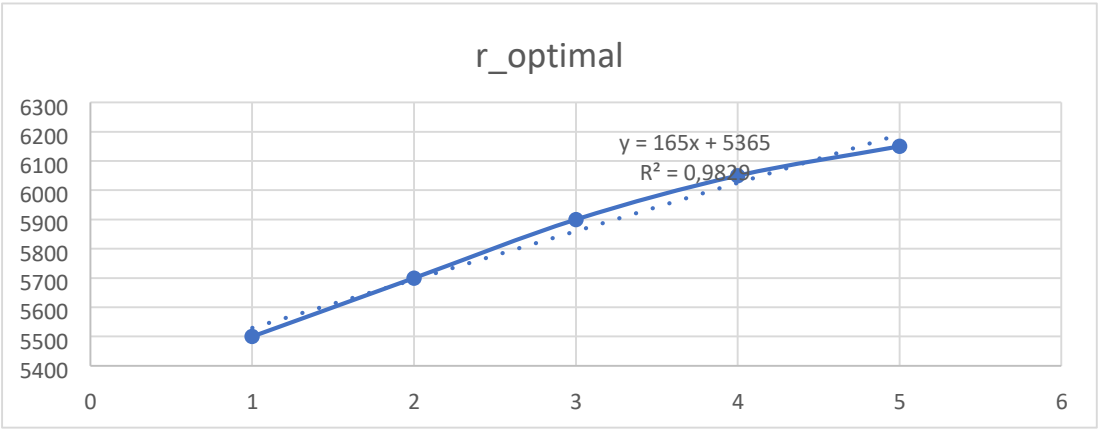


Figure 10 - Nuage de points entre i et le rayon optimal pour le dataset1

Nous avons donc choisi la valeur de i pour laquelle le système avait le potentiel maximal. On a donc tracé les courbes du F1-score et ROC pour les différents :

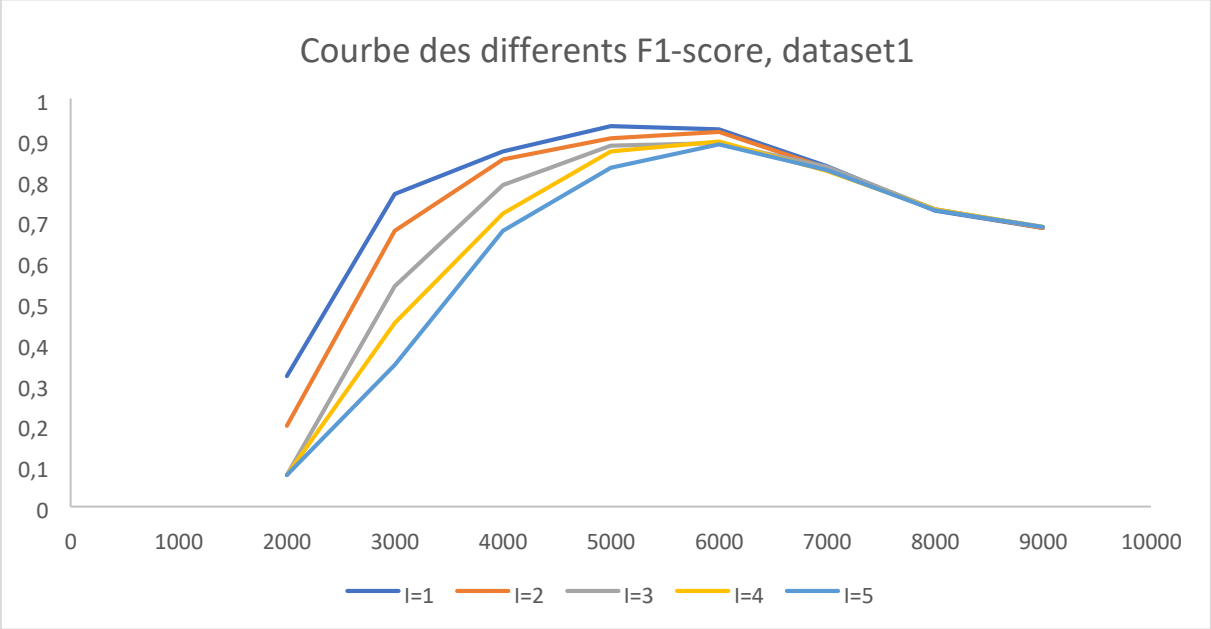


Figure 11- Courbes des F1-scores pour les différents i selon les rayons

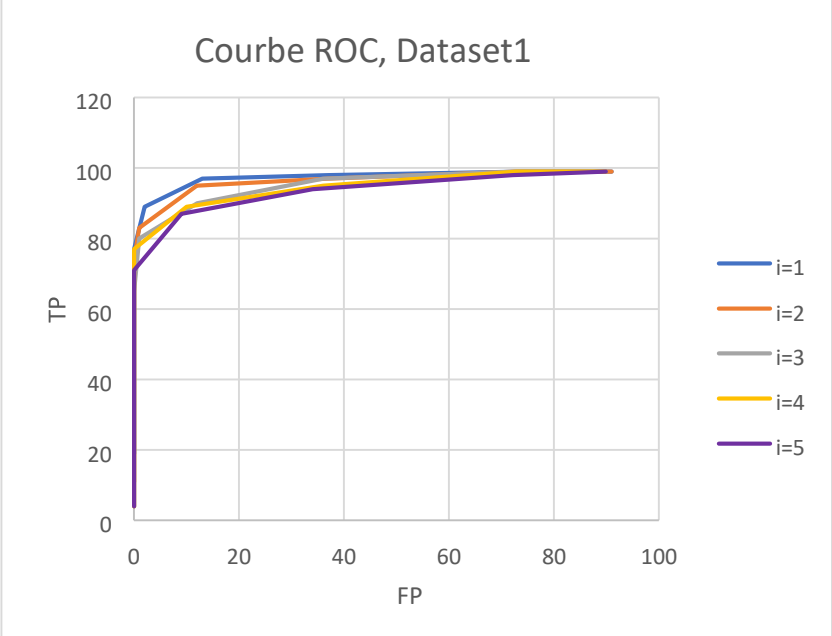


Figure 12 - Courbe ROC pour les i du dataset1

b) Pour le dataset2

On effectue les mêmes mesures cependant on ne prend que i= {1, 2, 3} car le jeu de données est moins fourni.

c) Valeurs retenues

	i	Rayon	Précision (%)	Exactitude (%)	Rappel (%)	Spécificité (%)	F1-score (%)
Dataset1	1	5500	98	96	94	98	96
Dataset2	1	7450	60	60	60	60	60

Le dataset2 est de mauvaise qualité, on l'observe en scrollant les photos :

- Beaucoup sont mal cadrées ;
- Le visage est de profil, trois-quarts face etc ;
- Deux photos uniquement par individu pour des individus ;
- D'autres sourient une photo sur deux (donc dents blanches au lieu des lèvres sombres) ;
  Des accessoires peuvent être portés comme des lunettes ;

3. Cas du Eigenfaces

Pour l'Eigenfaces il existe trois paramètres que l'on peut faire varier :

- Le rayon r
- Le nombre de composantes à garder k lors de l'ACP
- Le nombre de photo pour autoriser l'accès i

Pour faciliter les calculs et au vu des résultats précédents, on suppose que le système est le plus performant dans le cas où est fixé à la valeur 1.

a) Premier jeu de données

On a donc cherché le rayon optimal pour chaque k. De la même manière que pour la force brute, le rayon optimal se trouve par résolution graphique :

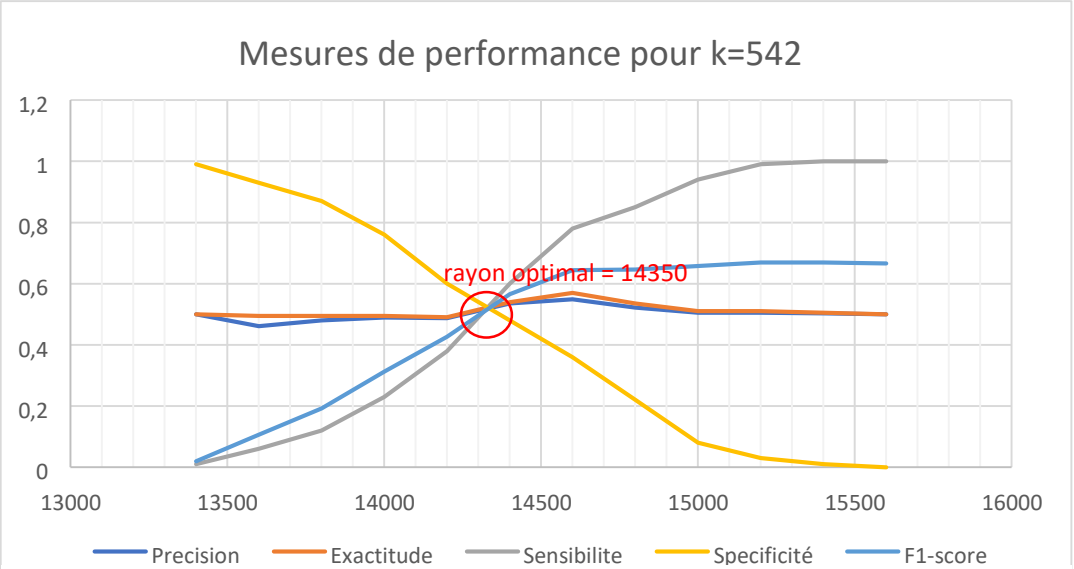


Figure 13- Mesures de performances pour K=542 et le dataset1

Nous avons également constaté une relation linéaire entre k et le rayon optimal :

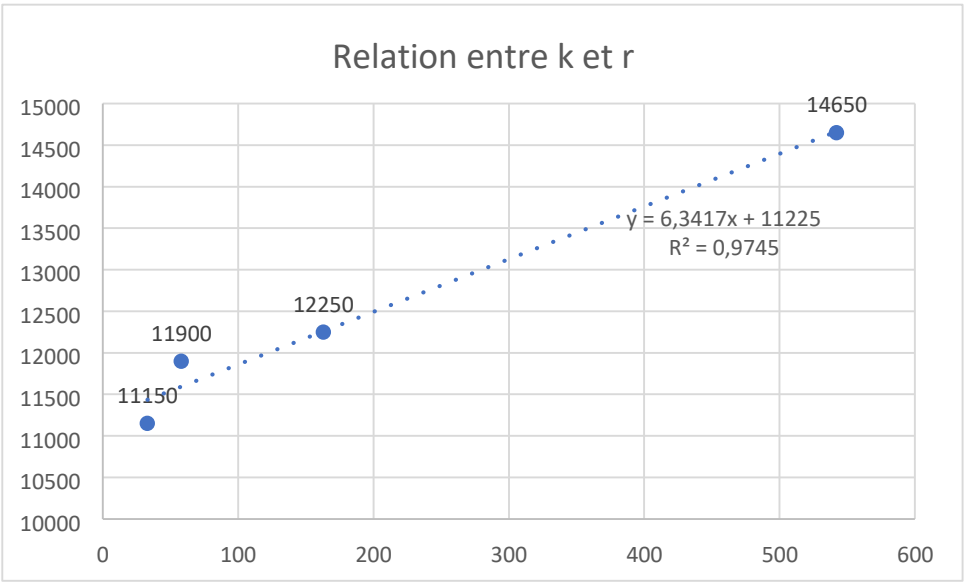


Figure 14 - Nuage de points entre k et r pour le dataset1

Cependant en traçant les Courbes ROC et en regardant les mesures de performances pour chaque k, on s'aperçoit que la performance du système semble indépendante de k :

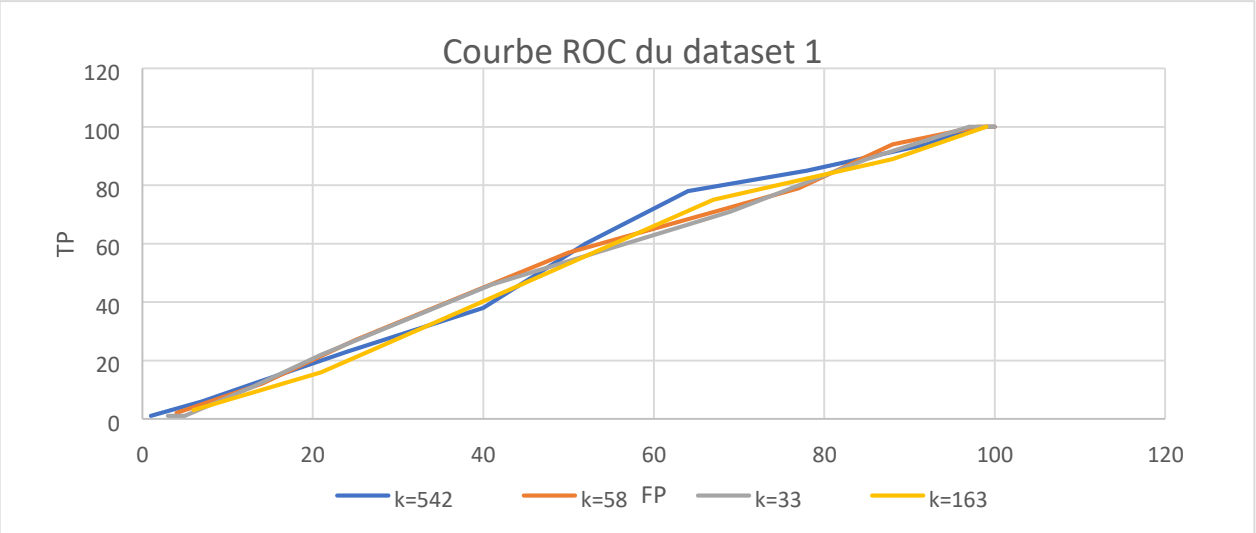


Figure 15 - Courbes ROC du dataset1 selon plusieurs k

	Précision	Exactitude	Rappel	Sensibilité	F1-score
K=33	0,51	0,51	0,85	0,17	0,63
K=58	0,50	0,50	0,79	0,21	0,61
K=163	0,51	0,51	0,92	0,1	0,65
K=542	0,52	0,54	0,87	0,20	0,65

Comme les valeurs sont à peu près similaires, on choisit de prendre k=163 car le F1-score est le plus élevé pour un temps d'exécution optimal (moins de projection à faire qu'avec K=542). Nous avons pris r=12558.

b) Pour le 2nd jeu de données

On obtient de la même manière le tableau suivant :

Nombre d'axes conservés	Rayon optimal	Précision	Exactitude	Rappel	Spécificité	F1-score
K=803	21825	0,49	0,48	0,9	0,05	0,63
K=167	13050	0,52	0,53	0,69	0,37	0,59
K=53	11170	0,51	0,51	0,74	0,28	0,60
K=30	10820	0,51	0,52	0,75	0,28	0,61

Pour les mêmes raisons qu'avec le premier jeu de données, on choisit k=167 qui est un bon compromis entre la qualité du système et la vitesse des requêtes.

V. Analyse de l'ACP

Dans cette partie nous allons analyser quelles photos contribuent le plus à un axe pour déterminer ce qu'il représente. Nous allons considérer les deux premiers axes pour rechercher les photos y contribuant le plus à l'aide du nuage de points dans le plan factoriel.

1. Pour le dataset1

a) Axe 1

Les photos qui contribuent le plus négativement à l'axe 1 sont : michael.11, anonym2.14, shamilc.15, fordj.10.

Les photos qui contribuent le plus positivement à l'axe 1 sont : dhaydo.8, rjwils.16, nmakri.7, pwest.6

Elles sont représentées ci-dessous :

Figure 16 - Photos contribuant négativement à l'axe 1

Figure 17 - Photos contribuant le plus positivement à l'axe 1

On remarque que les personnes du haut ont tous de la barbe ou bas du visage foncé et sont chauve ou ont le haut du visage clair. A l'inverse les personnes du bas ont le bas du visage clair et le haut foncé (cheveux ou casquette). On peut donc déduire que l'axe 1 représente la différence d'intensité de gris entre la partie supérieure et la partie inférieure de l'image.

b) Axe 2

De la même manière pour l'axe 2, les photos contribuant le négativement sont : tthurs.14, jbgood.1, 9556273.13, cprice7.

Et les photos contribuant le plus positivement sont : sirmcb.2, cladam.5, kbartl.7, gsmall.5.

Voici leurs représentations :

Figure 18 - Photos contribuant le plus négativement à l'axe 2

Figure 7 - Photos contribuant le plus positivement à l'axe 2

Ici, on voit bien que l'axe négatif est porté par des photos où le visage est masqué par des cheveux dans les coins. Positivement, les photos extraites mettent en avant des personnes ayant les coins supérieurs du visage dégarnis.

DATASET1	NEGATIF	POSITIF
AXE 1	Bas de l'image foncé / Haut clair	Haut de l'image foncé / Bas clair
AXE 2	Coins supérieurs foncés	Coins supérieurs clairs

2. Pour le dataset2

On effectue la même démarche.

a) Axe 1

Les photos qui contribuent le plus négativement sont : 0126.3, 0126.1, 0086.2, 0126.0

Les photos qui contribuent le plus positivement sont : 0976.2, 0974.2, 0968.1, 0977.3

Figure 19 - Photos contribuant le plus négativement à l'axe 1

Figure 20 - Photos contribuant le plus positivement à l'axe1

On observe que le côté négatif met en avant des photos ou la partie gauche de l'image est claire et la partie droite est foncée et inversement pour le côté positif.

b) Axe 2

Les photos qui contribuent le plus négativement sont : 0647.1, 0505.0, 0647.3, 0366.1

Les photos qui contribuent le plus positivement sont : 0273.1, 0885.0, 0768.7, 0885.1

Figure 21 - Photos contribuant le plus négativement à l'axe 2

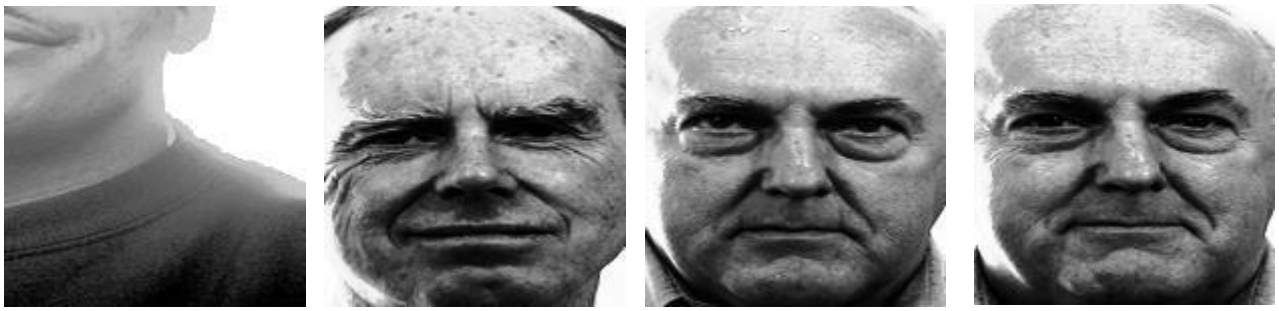


Figure 22 - Photos contribuant le plus positivement à l'axe 2

Idem que pour l'axe 1 du dataset1, on observe une opposition des nuances de gris entre les parties supérieures et inferieures des photos.

DATASET2	NEGATIVEMENT	POSITIVEMENT
AXE 1	Gauche claire / Droite foncée	Gauche foncée / Droite claire
AXE 2	Haut Foncé / Bas Clair	Bas Foncé / Haut Clair

VI. Comparatif Force Brute et Eigenfaces

1. Pour les mesures de performances

Il est indéniable que la recherche par force brute est bien plus efficace que la méthode Eigenfaces en termes de mesure de performance. La force brute a des taux proches de la perfection pour le 1er jeu de données qu'on ne retrouve pas l'Eigenfaces. Pour le second, comme les données sont de moins qualités les résultats en termes de performances sont plutôt mauvais, mais sont plus ou moins semblables.

2. Pour la vitesse d'exécution

Pour les vitesses d'exécution on obtient les temps suivants :

Type	Force Brute	ACP k=542	ACP k=163	ACP k=58	ACP k=33
Temps (s)	0.82	2.34	0.55	0.38	0.24

Nous avons évidemment une force brute pus longue à être exécutée qu'une ACP. Cependant, il ne faut pas choisir un k trop élevé car le temps d'exécution peut devenir plus long pour l'ACP qu'avec la force brute.

VII. Conclusion

Pour Conclure, nous avons vu deux méthodes d'authentification faciales : la recherche par force brute et la méthode des Eigenfaces. Chaque méthode a ses avantages et ses inconvénients, l'une est plus rapide mais moins précise et l'autre plus précise mais plus lente. On peut dire qu'une méthode serait plus efficace pour un cas précis. Par exemple, pour déverrouiller son téléphone où il faut un temps minime la méthode des Eigenfaces semble appropriée. A l'inverse, pour accéder aux codes de la bombe nucléaire, il est plus intéressant de se tourner vers une méthode de recherche par force brute.

Pour aller plus loin, il aurait été intéressant de nettoyer le jeu de données en enlevant les photos mal cadrées. De plus, pour la méthode Eigenfaces, comme les requêtes sont assez rapides, on peut en envoyer plusieurs sans que le système soit fortement ralenti. Ainsi lorsque l'on est devant un capteur d'authentification faciale, plusieurs images sont prises et non une seule comme dans notre programme.

VIII. Bibliographie

http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_Nb_Components_PCA.pdf

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

https://fr.wikipedia.org/wiki/Courbe_ROC https://en.wikipedia.org/wiki/F1_score

https://pythonhow.com/measure-execution-time-python-code/