ADRIEN LAVILLONNIERE - HIEN MINH NGUYEN

RAPPORT DU PROJET DE BIGDATA, PARTIE 1

Polytech Paris-Sud, ET5 Informatique, 2018

I - Introduction

L'objectif de ce projet est de créer un programme capable de classer des images de nombres, issues du monde réel.

Nous utiliserons la bibliothèque SVHN, qui contient environ 600 000 numéros de maison extraits de Google Street View. Nous n'en utiliserons que 100 000 environ afin d'entraîner notre modèle et le tester. L'intérêt de SVHN comparé à d'autre datasets repose principalement sur la quantité très élevée de données issues du monde réel. Ces données possèdent toutes un label correspondant à chaque chiffre présent sur une image.

Dans un premier temps, nous utiliserons plusieurs algorithmes de classification tels que le classifieur à distance minimum (DMIN), les classifieurs de Scikit-Learn (SVM). Ces méthodes seront assistées d'un traitement préalable des données (pré-processing), ainsi qu'une réduction de la dimension des vecteurs (PCA). Dans un deuxième temps, nous utiliserons un réseau de neurones convolutionnel pour effectuer cette classification.

Ce rapport traitera exclusivement de la première partie du projet. À travers l'expérimentation de plusieurs méthodes de pré-processing et de classification, nous avons entraîné et amélioré notre modèle, jusqu'à atteindre autour de 18% de réussite avec DMIN et pré-processing, autour de 16% pour C-Support Vector Classification (SVC) et 46% pour le KNeighborsClassifier (KNN).

Ce projet nous a permis d'expérimenter avec les bases du machine learning afin de gérer un gros volume de données. En effet, les classificateurs d'images sont une application répandue et très utile au monde des big data de nos jours. La reconnaissance de nombres est une étape basique mais nécessaire dans de nombreux domaines, comme la surveillance, la robotique ou encore la réalité augmentée.

II - Déroulement du projet

Afin de faire reconnaître un chiffre par la machine, il existe plusieurs solutions. Dans ce projet, nous allons implémenter trois classificateurs : DMIN, un classifieur parmis les Support Vector Machines (SCM) et un autre parmis les plus proches voisins.

Quant au PCA, il optimisera les données en entrée des classificateurs, en réduisant leur dimension et permettra en principe de diminuer les temps de calcul.

2.1 Pré-processing

Les images non traitées sont insuffisamment claires et nettes pour pouvoir faire une différence entre chaque chiffre. Voici un exemple d'image non traitée :

Nous devons donc les faire passer par un pré-processing, c'est-à-dire des algorithmes qui modifieront chaque pixel d'une image pour la rendre plus facile à traiter pour le classificateur.

Nous avons implémenté plusieurs types de pré-processing : passer l'image en négatif, en noir et blanc, appliquer un seuillage d'image, un filtre passe-haut, un filtre passe-bas, un gradient, un filtre de Sobel, augmenter la luminosité, le contraste et l'égalisation par histogramme. Ces filtres possèdent des paramètres réglables ou non.

Voici un exemple d'image traitée par le filtre *thresholdingV2* :

Nous avons également réfléchi à un moyen de sauvegarder les images dans un fichier après leur avoir appliqué un pré-processing. En effet, le temps de calcul requis pour appliquer un filtre sur 1000 images est de l'ordre d'une minute (cela dépend de chaque filtre), nous ne pouvons donc pas nous permettre de répéter le pré-processing sur 100 000 images à chaque lancement du programme.

Nous avons donc utilisé la librairie Pickle, qui permet de sauvegarder et de charger des variables de manière transparente. Les fichiers en sortie pour chaque pré-processing font plus de 230 Mo, ce qui est une taille similaire à la taille des jeux de tests originaux.

Cette librairie est ensuite réutilisé pour sauvegarder les données après avoir appliqué le PCA ou encore pour sauvegarder un modèle sklearn.

2.2 Classifieur à Distance Minimum

La solution la plus basique pour classifier des images consiste à donner à la machine en entrée un nombre conséquent d'images de chiffres labellisés, calculer le vecteur distance minimal pour chaque nombre (chaque nombre correspond à une classe). Notre ensemble de données de test peut être divisé en 10 classes, une pour chaque chiffre de 0 à 9. Afin d'entraîner notre modèle, nous déterminons le vecteur moyen de distance minimal pour chacune de ces classes. Ensuite, nous pouvons nous servir de ce modèle, le comparer avec le vecteur d'une image de chiffre inconnu et ainsi lui attribuer une classe.

Un problème d'optimisation a été relevé : afin de calculer les vecteurs pour chaque classe, il fallait plusieurs secondes (entre 2 et 4 secondes selon nos observations). Cela est dû au nombre conséquent d'images en entrée et à leur chargement en mémoire. Ce temps de calcul semble négligeable par rapport au temps requis par le pré-processing, de l'ordre d'une heure dans le pire des cas.

2.3 Analyse en Composantes Principales

L'ACP (Analyse en Composantes Principales) est une technique permettant de réduire la dimension des vecteurs de notre dataset. De dimension d (correspondant aux coordonnées de l'image), le vecteur originel sera projeté sur un sous-espace de dimension p. Cela permettra d'améliorer les calculs en réduisant le nombre d'axes, tout en gardant le maximum d'information.

Pour réaliser l'ACP, on a utilisé la bibliothèque Scikit-Learn. Pour utiliser la méthode sklearn.decomposition.PCA, il faut au-préalable modifier la structure des données. En effet, les tableaux train_32x32['X'] et test_32x32['X'] sont de dimensions 4, ce qui est incompatible avec la méthode. Il faut alors réduire les tableaux de sorte à avoir des tableaux de dimensions 2. Pour ne pas perdre d'informations, on se retrouve finalement avec un tableau de vecteur de dimension 32x32x3 (chaque vecteur représentant une image).

PCA possède une option permettant de définir le nombre de composants suite à son application. Dans la suite des expérimentations, on utilise cette option pour définir le pourcentage de composants que l'on souhaite conserver.

2.4 SVM et Plus Proche Voisin

Enfin, nous avons implémenté un classifieur de type Support Vector Machines. Les SVM sont des méthodes de classification, implémentées dans la librairie Scikit-learn. Elles sont efficaces pour les données à haute dimension.

La méthode SVC possède de nombreuses options pour pouvoir s'adapter aux données comme par exemple le choix du kernel ou bien la possibilité de d'associer un poids à chaque classe pour les problèmes déséquilibrés. On a choisi d'utiliser SCV avec le kernel mis sur "linear". Cette méthode implémente la stratégie "one-vs-one". C'est-à-dire que l'ordre de comparaison pour des classes allant de 0 à n "0 vs 1", "0 vs 2" , … "0 vs n", "1 vs 2", "1 vs 3", "1 vs n", . . . "n-1 vs n".

Le classifieur du plus proche voisin se fait grâce à KNeighborsClassifier. Cette méthode implémente la méthode d'apprentissage supervisé des k plus proches voisins. Elle implémente une option permettant de définir le nombre de voisins, qui va nous servir pour faire nos expérimentations.

III - Résultats

3.1 Preprocessing avec DMIN

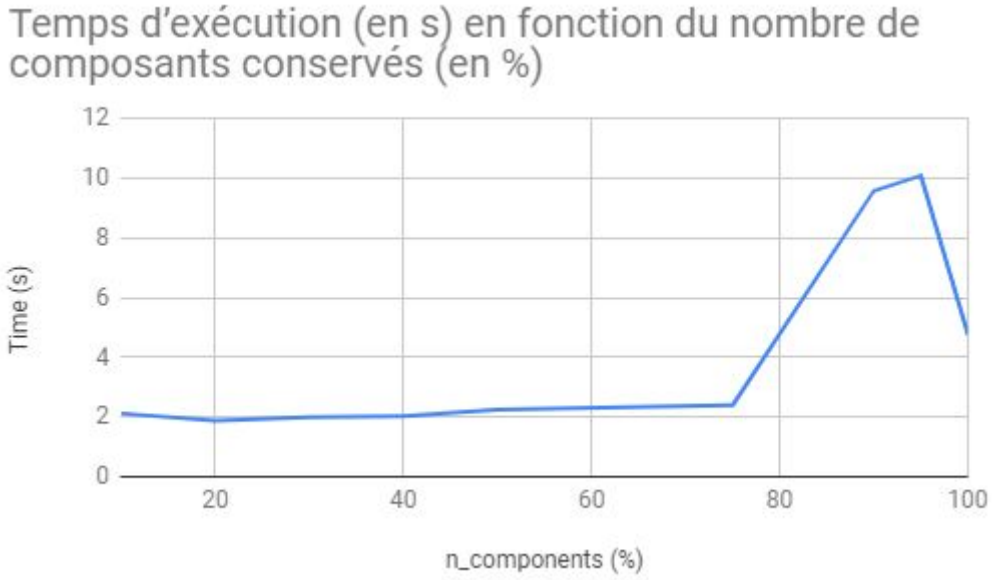
Plusieurs preprocessing ont été réalisé dans le but d'obtenir la meilleur performance possible avec le DMIN. Chaque preprocessing est une combinaison de filtre appliquée à l'ensemble des images. Le tableau suivant résume les temps d'exécution, ainsi que les taux d'erreur sur les jeux d'entraînement et de test.

Succès (en %)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
gradient + highpass	9.37	19.60	15.88	11.91	9.33	10.78	6.55	8.45	5.94	8.95	12.34
gradient + lowpass	5.61	20.46	14.6	10.01	10.87	11.00	7.31	10.77	5.50	11.57	12.24
negative + lowpass	6.51	19.85	17.13	11.78	10.77	9.66	6.00	6.25	6.51	06.02	11.88
negative + highpass	6.29	21.25	18.33	13.10	25.88	8,04	16.94	6.28	6.81	10.86	8.92

3.2 ACP avec DMIN

Pour la suite des expérimentations, le preprocessing appliqué est le filtre *thresholdingV2*.

Si on ne prends pas en compte le temps que prends l'ACP pour s'effectuer, alors on remarque un net gain sur le temps d'exécution du DMIN. Cependant, la performance est nettement réduite. On passe d'une précision moyenne de 12% à 4%. Le graphique 1 ci-dessous montre l'impact de la réduction de dimension sur le temps d'exécution du DMIN en fonction de la dimension de l'ACP.

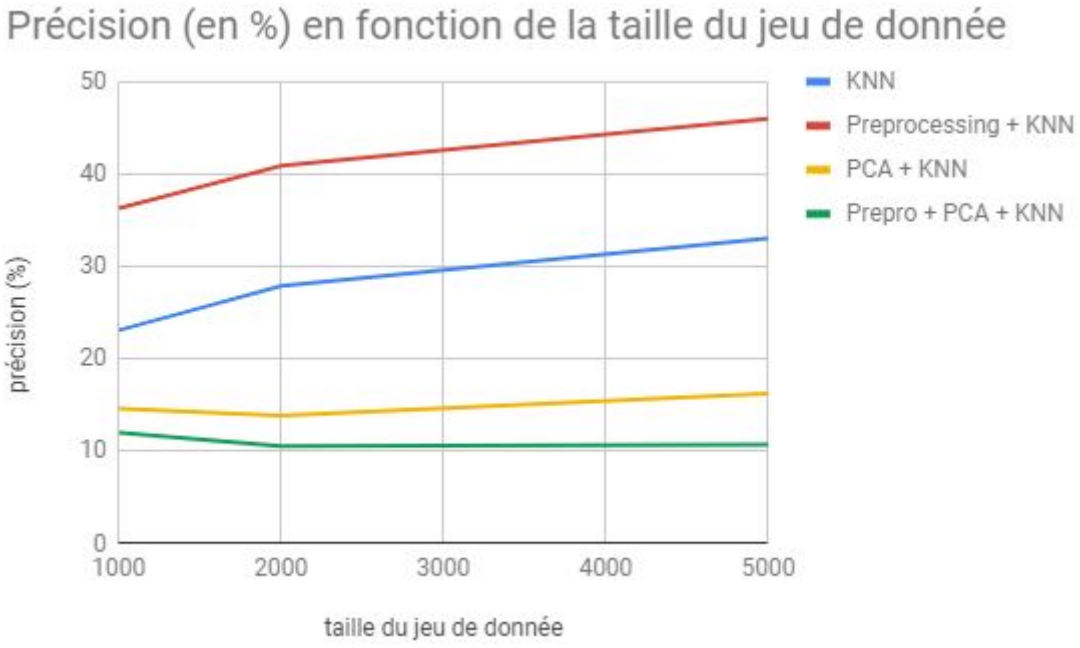


graphique 1

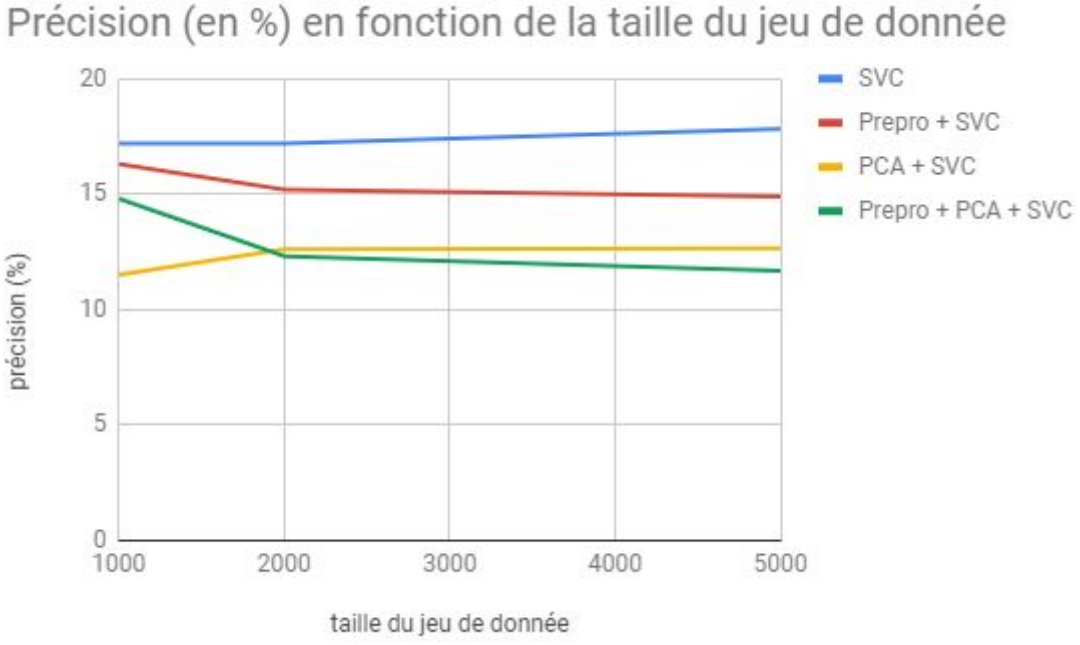
3.3 Scikit-Learn

Les résultats obtenus pour le SVC sont proche des résultats du DMIN tandis que ceux du KNN sont deux fois plus élevés. Une cause probable de ce résultat est la phase de pré-processing, qui est peu adaptée au SVC. En effet, alors que la configuration preprocessing + KNN augmente les performances, la configuration preprocessing + SVC la diminue.

En ce qui concerne les paramètres du KNN, on remarque vite que la taille du jeu de donnée joue beaucoup sur la performance (cf graphique 2). Cette tendance est également vrai pour le SVC (cf graphique 3).

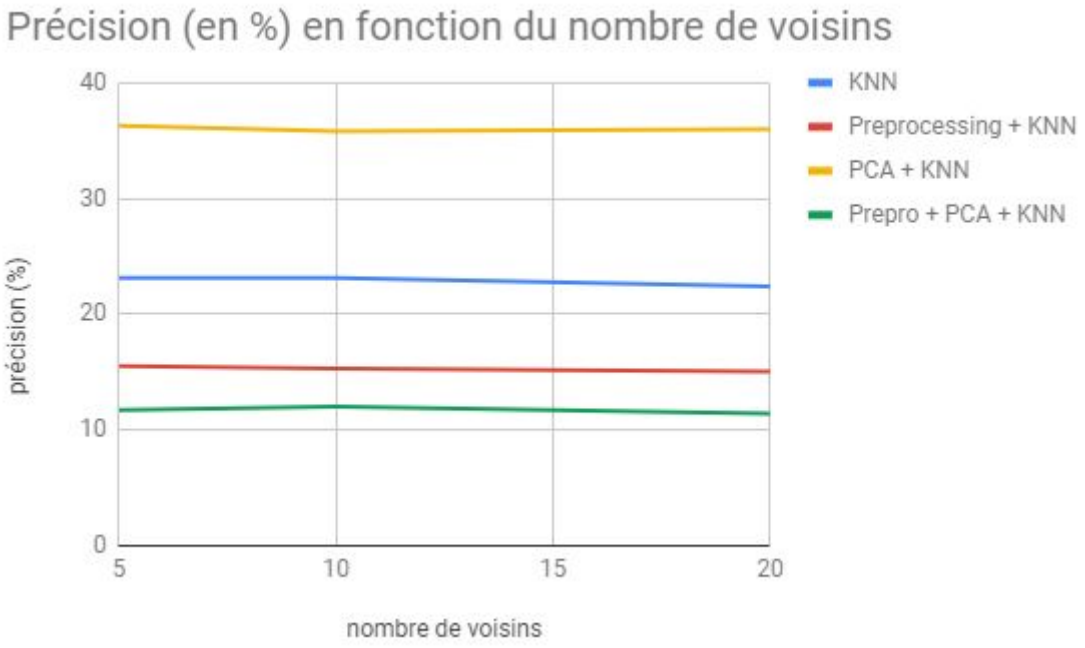


graphique 2



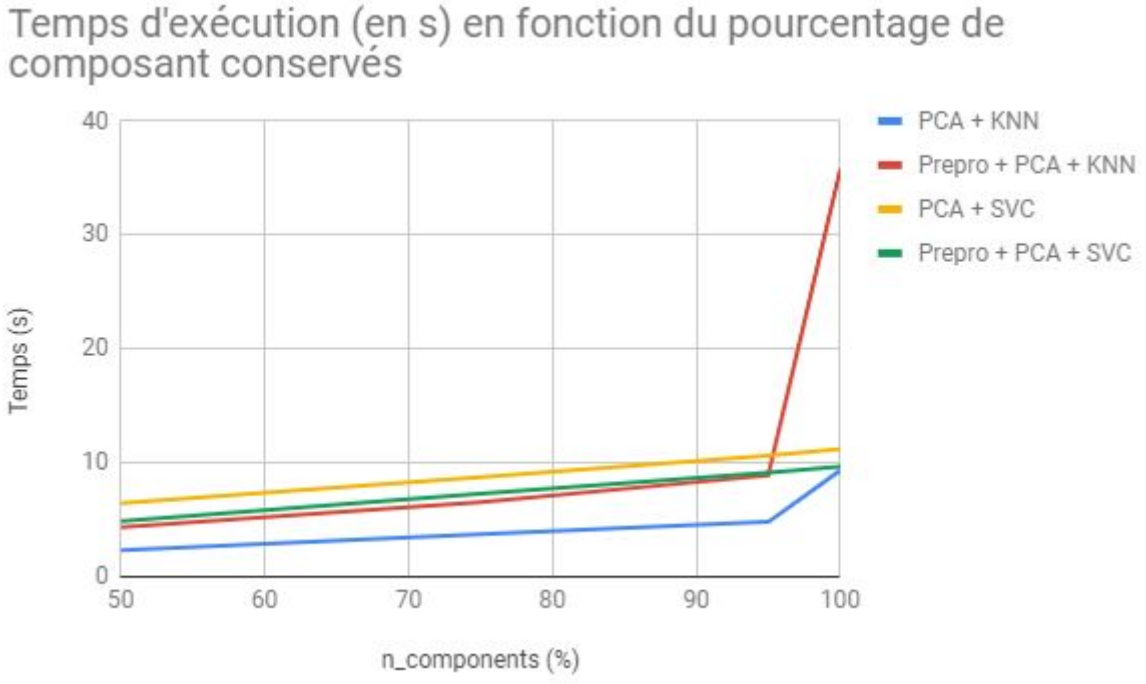
graphique 3

Le paramètre k de la méthode des k plus proches voisins joue très peu sur la performance dans les configurations testées.Cependant, une légère baisse des performances est observée lorsque k devient trop grand.



graphique 4

L'APC, quant à lui, améliore toujours le temps d'exécution du classifieur mais son propre temps d'exécution le rend rarement rentable pour la perte de performance.



graphique 5

Les matrices de confusions suivantes représentent les meilleurs résultats obtenus pour chaque classifieurs. On peut constater que les chiffres 0, 1 et 3 ont globalement de bon résultat alors qu'à l'inverse, les chiffres 6, 7, 8 et 9 ont globalement de mauvais résultat.

On peut interpréter cette tendance comme un gage de la mauvaise qualité des images ayant pour label 6, 7, 8 et 9 et la bonne qualité des images avec pour label 0, 1 et 3. Mais on peut aussi interpréter ces résultats comme la ressemblance du 7 et du 1, du 6 et du 9 ou du 0 et du 8. Cette ressemblance pourrait être la cause des mauvais résultats.

Pour éviter ce problème, il est possible d'améliorer la phase de pré-processing mais aussi de modifier les images pour accentuer les distinctions entre les chiffres, en ajoutant une barre horizontale au 7 par exemple.

Preprocessing + KNN (5 voisins, 5000 images) :

46,02%	0	1	2	3	4	5	6	7	8	9
0	844	28	33	29	11	6	15	9	6	7
1	248	365	62	19	11	7	25	21	10	7
2	232	57	173	46	19	6	15	20	6	17
3	115	16	12	303	11	11	0	7	8	13
4	124	41	50	33	101	40	4	28	13	16
5	80	24	22	81	31	77	0	28	8	30
6	145	41	13	5	10	1	148	2	6	5
7	70	30	32	20	19	52	1	58	10	24
8	72	39	17	15	21	11	7	25	71	40
9	42	19	12	32	17	19	3	11	23	161

matrice de confusion 1

SVC (5000 images) :

17,82%	0	1	2	3	4	5	6	7	8	9
0	313	169	90	98	82	64	51	50	31	40
1	171	155	78	83	68	48	57	45	58	22
2	118	74	99	52	48	38	47	27	27	31
3	105	77	40	83	36	30	35	29	28	33
4	72	66	73	49	54	25	22	46	22	21
5	76	60	40	38	30	54	21	27	14	21
6	66	70	37	52	35	25	42	15	24	10
7	53	42	42	28	28	44	25	26	12	16
8	57	39	54	35	23	29	24	15	37	14
9	69	52	36	41	31	25	23	20	14	28

matrice de confusion 2

IV - Commentaires

L'élément principal permettant d'expliquer les mauvais résultats du classificateur à distance minimal ainsi que du SVM est sans doute le pré-processing. Plusieurs améliorations sur celui-ci peuvent être effectuées.

Il semble utile de noter qu'une image identifiable par l'humain n'est pas forcément plus facile à traiter par la machine : il ne faut donc pas s'appuyer que sur des résultats visuels sur quelques images tests pour déterminer la qualité d'un pré-processing, mais bien de sa performance dans des conditions réelles.

Tout d'abord, les chiffres sont placés de manière aléatoires sur l'image. Leur taille et couleur varient également grandement. À partir des distances minimales, on ne peut pas donc générer un modèle précis pour chaque chiffre. On obtient plutôt un modèle flou, avec vaguement la forme du chiffre en question au centre de l'image.

De plus, les chiffres ne sont pas seuls sur leur images. En effet, il y a beaucoup d'éléments présents sur les images qui peuvent perturber le processus d'entraînement, comme les rebords de la plaque contenant le chiffre. Nous avons également remarqué que

certaines images contenaient des paires de chiffres : cela brouille encore plus les pistes et rend certaines images inutilisables.

Afin de perfectionner la classification, il faudrait un algorithme permettant de recentrer et de redimensionner chaque chiffre au centre de l'image. [1]

Dans certains cas, la qualité de certaines images est très faible et l'oeil humain ne parvient pas à distinguer le chiffre. Le trop grand flou de l'image influence grandement la qualité de l'apprentissage : au final, certaines images auxquelles nous avons appliqué le préprocessing deviennent quasiment unicolores.

Afin d'améliorer la classification, des améliorations peuvent être réalisées sur les points ci-dessus, mais également sur les algorithmes d'apprentissage. En effet, le machine learning ne peut produire qu'un modèle aux capacités limitées. Un apprentissage de type *machine learning* serait bien plus optimisé et efficace : par exemple, nous pourrions reconnaître les caractéristiques des chiffres, les décomposer…, ou passer l'image dans un réseau de neurone composé de plusieurs couches. C'est précisément ce que nous allons faire pour la deuxième partie de ce projet, en utilisant la librairie PyTorch.

V - Utilisation de classif.py

L'exécutable est classif.py. Le programme requiert numpy et scipy.io. Attention, le programme charge un modèle à partir d'un chemin fixe. Le fichier train_32x32.mat doit être dans le même dossier pour exécuter le .py. Ce fichier doit être ajouter à la main, car sinon l'archive zip est trop lourde pour l'espace Dokeos.

On peut passer plusieurs options :

- -h (ou --help) : aide
- -d (ou --data) : chemin du fichier .mat
- -s (ou --size) : taille maximale à analyser

[1] https://github.com/thomalm/svhn-multi-digit/blob/master/05-svhn-multi-preprocessing.ipynb