

tp Python – Machine Learning

Marc-antoine auvray – PAUL delamarre – Matthieu le gall – Nathan papet

Détection du type de cartes de séjour

ecole D’ingenieur du cnam – biG DATA ET IA – SEMESTRE 3 2022-2023

S3 2022

Table des matières

[Introduction et problématique 2](#_Toc120887628)

[1. Présentation des données 4](#_Toc120887629)

[*2.* *Machine Learning* 5](#_Toc120887630)

[2.1. Introduction 5](#_Toc120887631)

[*2.1.1.* *Data Preparation* 5](#_Toc120887632)

[*2.1.2.* *Cross-Validation* 5](#_Toc120887633)

[2.2. Modèle introductif : régression logistique 6](#_Toc120887634)

[2.3. Vingt-sept modèles 9](#_Toc120887635)

[3. *Deep Learning* 10](#_Toc120887636)

[4. Réflexion et pistes d’amélioration 10](#_Toc120887637)

[5. Conclusion 10](#_Toc120887638)

[Bibliographie 10](#_Toc120887639)

# Introduction et problématique

Automatiser les processus de tâches répétitives telles que la reconnaissance automatique de documents est l’un des enjeux majeurs pour beaucoup d’entreprises ou autres structures. Cela peut notamment offrir aux équipes plus de confort pendant l’utilisation de ces processus ou encore plus de facilité quant à la compréhension du fonctionnement et de l’objectif de ces processus. Ces processus automatisés étant plus efficients, cela apporte alors un gain de temps, et en conséquence, les équipes peuvent investir ce temps pour travailler sur d’autres tâches notamment certaines qui seraient plus avancées.

Parmi les documents qu’il peut être utile de classer de manière automatique dans un contexte d’entreprise ou plus souvent d’administration publique, on retrouve les titres de séjour. Ces cartes sont nécessaires pour les ressortissants étrangers (hors Union Européenne) souhaitant résider en France pour une période supérieure à trois mois. Ces titres de séjour ont changé d’aspect en 2020 pour passer d’une carte rose à une carte blanche. Certaines informations ont également changé.

Une image contenant texte, journal

Description générée automatiquement

Figure 2 - Ancienne carte de séjour

Figure 1 - Nouvelle carte de séjour

Voici un exemple des deux types de carte de séjour. A gauche une nouvelle carte de séjour et à droite une ancienne carte de séjour.

Nous allons donc chercher à automatiser le processus de catégorisation de ces deux types de cartes. Le but de ce projet n’est pas de trouver une solution parfaite. La motivation est plus d’ordre exploratoire. En l’occurrence, plusieurs notions autour de l’intelligence artificielle seront travaillées sur *Python* :

-traitement et préparation de données ;

-*machine learning*;

-*deep learning*;

-*cross-validation* ;

-classification supervisée ;

-réseaux de neurones ;

-*data augmentation*.

Dans ce rapport nous détaillerons donc notre démarche, les problèmes rencontrés ainsi que les solutions proposées pour répondre à ces problèmes. Nous présenterons comment nous avons issu les données des images pour pouvoir les traiter et s’en servir, et les méthodes utilisées avec ces données pour prédire le type d’une carte.

# Etat des lieux

## Présentation des données

Pour travailler sur ce sujet, nous n’avons pas eu de jeu de données fourni. En effet, étant donné les données personnelles présentes sur ces cartes, il est difficile de trouver un jeu de données public sur ce thème. Ainsi, nous avons dû constituer notre base de données grâce à des recherches internet via Google Images. Celle-ci est constituée d’images qui peuvent être de deux types différents de cartes de séjours :

-les nouvelles cartes de séjours ;

-les anciennes cartes de séjours.

Lors de nos recherches, nous avons été confrontés à plusieurs problèmes. Certaines images étaient erronées ou encore floutées laissant des informations partielles ou inexistantes. La qualité de beaucoup d’images étaient basses et donc non exploitables.

Pour finir, nous avons pu avoir un jeu de données de 28 images correctes : 16 anciennes cartes et 12 nouvelles cartes. Les nouvelles cartes étaient plus difficiles à trouver.

Une fois ce premier travail effectué nous avons pu passer sous *Python*.

## Outils *Python*

### Fichiers *pickle*

### Package 1

# *Machine Learning*

## Introduction

### *Data Preparation*

Pour prédire une variable binaire en utilisant des algorithmes de *machine learning* sous *Python*, ceux-ci ont besoin de variables explicatives numériques en entrée. Il faut donc trouver le moyen de résumer chaque image en quelques valeurs la décrivant : l’étape de *data preparation*. Nous nous sommes donc concentrés sur les colorations RGB des images (Rouge – Vert – Bleu).

Chaque image a un certain nombre de pixels (longueur de l’image × largeur de l’image). Chaque pixel a une valeur de rouge, une valeur de vert et une valeur de bleu. Pour chaque image, on peut donc faire la moyenne de tous les rouges, la moyenne de tous les bleus et la moyenne de tous les verts. On arrive donc à un tableau à 4 colonnes, avec pour chaque image sa catégorie (0/1, soit ancien/nouveau type) et trois valeurs numériques décrivant sa coloration. Nous avons alors eu l’idée de prendre la moyenne pour résumer chaque couleur primaire. On pourrait aussi choisir plutôt la médiane, puis comparer les résultats des deux.

Nous avons alors créé sur *Python* une classe nommé *ColorsData*. Les objectifs de cette classe sont de :

-pouvoir à partir de 2 dossiers avec des images, créer une base de données catégorisée binairement ;

-traduire les informations de fichiers images en informations numériques ;

-proposer de manière optionnelle des noms aux catégories ;

-ajouter à sa base de données des nouvelles images ;

-exporter la base de données vers un fichier *.pickle* ;

-importer une base de données depuis un fichier *.pickle*.

Elle a 6 attributs : *files\_path*, *cat\_paths*, *data\_file\_name*, *cat\_names*, *data* et *target*. L’attribut *files\_path* est un *string*, le chemin du projet, nous avons mis une valeur par défaut. L’attribut *cat\_paths* est une liste de 2 *strings* : chaque *string* correspond au chemin d’un dossier où sont les images d’une catégorie (sans le chemin entier du projet), nous avons mis des valeurs par défaut. L’attribut *data\_file\_name* est un *string* étant le nom d’un fichier *pickle* qui contient ou contiendra les données, il doit alors se finir par "*.pickle"*, nous avons mis une valeur par défaut. L’attribut *cat\_names* est une liste de 2 *string*, ils représentent des noms personnalisés décrivant chacune des 2 catégories. Il faut faire attention à ce que les chemins et les noms personnalisés correspondent bien. Nous lui avons mis des valeurs par défauts. Les attributs *data* et *target* sont des listes initialisées vides et eux ne sont pas obligatoires lors de la création d’un objet du type de cette classe. La première contiendra les données de variables explicatives et la deuxième la variable à expliquer.

La classe a 4 fonctions : *get\_one\_data*, *get\_one\_data\_and\_target*, *add\_image* et *get\_data\_and\_target*.

La fonction *get\_one\_data* prend en argument un fichier image ainsi qu’un chemin. Elle renvoie une liste de 3 valeurs numériques qui décrivent chacune une couleur primaire de l’image comme expliqué précédemment.

La fonction *get\_one\_data\_and\_target* prend aussi un fichier image en argument. Elle prend également un chemin mais celui-ci doit être un sous-chemin d’un dossier d’une catégorie, dans le dossier du projet. Elle utilise aussi les attributs *files\_*path et *cat\_paths* et la fonction *get\_one\_*data de la classe. Elle renvoie alors un tuple composé : d’une liste retournée grâce à la fonction *get\_one\_data* et d’une valeur (0 ou 1) décrivant la catégorie de l’image.

La fonction *add\_image* prend les mêmes arguments que la fonction précédente. Elle utilise directement les attributs *data* et *target* de la classe et la fonction *get\_one\_data\_and\_target*. Elle va récupérer le tuple que cette dernière retourne. Elle va alors ajouter le premier élément du tuple (la liste décrivant les couleurs de l’image) à la fin de l’attribut *data* de l’objet *ColorsData* et ajouter le deuxième élément (la valeur décrivant la catégorie de l’image) à la fin l’attribut *target*.

La fonction *get\_data\_and\_target* prend 2 arguments : un booléen nommé *update* par défaut à *False* et un *string* nommé *updated\_file\_name* dont nous avons choisi de mettre en valeur défaut la même valeur de l’attribut *data\_file\_name* de l’objet *ColorsData*. Elle peut utiliser directement les attributs de la classe *data*, *target*, *cat\_paths* et *data\_file\_name* de la classe et peut aussi utiliser la fonction *add\_image*. En fonction de la valeur de *update*, elle va faire 2 choses différentes. Le choix pour cette valeur est à faire en fonction de si on veut réactualiser les données. Si l’on souhaite réactualiser les données en déclarant un *update* *True*, alors cette fonction va prendre toutes les images issues des 2 dossiers de catégories et créer un fichier *pickle*. Ce fichier contiendra 2 listes de même taille : une liste avec pour chaque image une liste de 3 couleurs la décrivant, et une liste avec les numéros de catégorie de chacune des images. Si l’*update* reste *False*, alors c’est que l’on part du principe qu’il y a déjà un fichier *pickle* avec les données agencées comme décrit précédemment.

Etant donné que nous avons déjà créé un fichier *pickle* avec les données dedans lors de notre travail, il nous suffit de procéder aux étapes suivantes :

-initialisation d’un objet de type *ColorsData* en laissant les valeurs par défaut aux attributs ;

-application sur cet objet de la fonction *get\_data\_and\_target* sans nouvelles valeurs à la place des valeurs par défaut non plus ;

-extraction des attributs *data* et *target* de cet objet pour une utilisation dans un modèle de *machine learning*.

### *Cross-Validation*

Pour essayer de tester les performances d’un modèle de *machine learning*, il va falloir garder des données de côté dans un "jeu de test“ pour les tester sur un modèle entraîné sur les autres données, du "jeu d’entraînement" : l’étape de *cross-validation*.

Parmi les méthodes de *cross-validation* existantes, nous nous sommes intéressés au méthodes *LeaveOneOut* et *LeavePOut*, dont la documentation est disponible sur le site de *sklearn*.

https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.LeaveOneOut.html

https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.LeavePOut.html

La première permet de mettre une seule donnée dans le jeu de test et le reste dans le jeu d’entraînement. On peut ensuite faire autant de coupes et de modèles correspondants qu’il y a d’individus/de cartes. Quand l’échantillon est très petit, cette méthode est idéale et on peut la rendre facilement visualisable. La deuxième permet de mettre un nombre *p* de données dans le jeu de test et le reste dans le jeu d’entraînement. On peut aussi boucler pour créer autant de modèles qu’il existe de couples entraînement-test. Il faut aussi que le nombre de données soit petit.

Nous avons donc choisi de garder deux données (images) pour chaque jeu de test. Nous voulions néanmoins avoir une donnée par catégorie différente (un couple nouvelle carte – ancienne carte). Il n’existe pas de méthode toute faite par *sklearn*. Nous avons décidé de baptiser cette méthode "*LeaveTwoDiffOut*" (*leave two different data out*). Nous nous sommes aidés d’une aide proposée sur le forum de *stackoverflow*.

<https://stackoverflow.com/questions/63705004/how-to-split-data-with-leave-one-pair-out-cross-validation-leavepout-for-binar>

Nous avons alors créé la classe *LeaveTwoDiffOut* dont les objectifs sont les suivants :

-à partir d’un jeu de données et d’un modèle de *machine learning*, proposer toutes les coupes possibles selon la méthode décrite précédemment ;

-proposer ces coupes à travers une base de données dont chaque "individu" représenterait une coupe possible et différente des autres avec à l’intérieur précisées : les données explicatives du jeu de d’entrainement de la coupe, les données à expliquer de son jeu d’entrainement, les données explicatives de son jeu de test, les données à expliquer de son jeu de test et les données du jeu de test prédites par le modèle donné ;

-proposer la comparaison (d’égalité) entre les données à expliquer du jeu de test et les prédictions ;

-proposer cette comparaison à travers un texte qui la décrit de manière résumée ;

-représenter cette comparaison à travers un camembert en 2 parties (les prédictions réussies et les prédictions ratées).

La classe *LeaveTwoDiffOut* a quatre attributs obligatoires lors de la création d’un objet de cette classe : *model*, *data*, *target* et *cat\_names*.

## Modèle introductif : régression logistique

Au cours de cette première partie, nous avons entrainé un modèle de régression logistique.

* Apprentissage du modèle :

Ce modèle étant introductif à ce projet, celui choisi est un modèle de régression logistique. Il est idéal pour prédire des données binaires. L’un des modèles linéaires traduisant une régression logistique entraînée sur *Python* est le suivant :

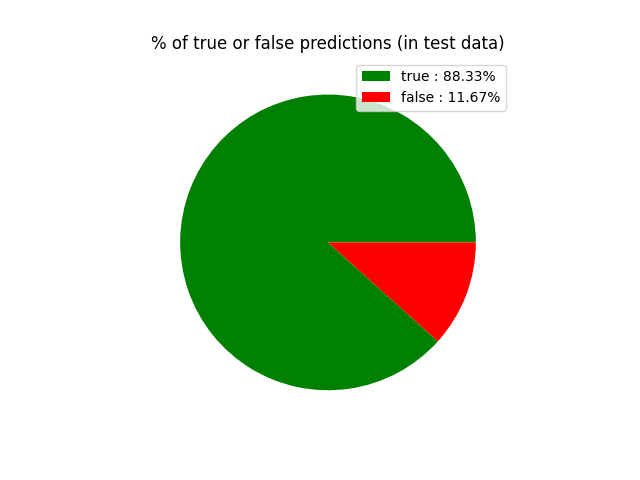
-0.74174263-0.39915359\*R + 1.16623781\*G +-0.75074005\*B

Exemples :

Pour l’image de catégorie 0/ancienne ayant pour valeurs R, G, B = [205, 192, 214] et pour celle de catégorie 1/nouvelle ayant pour valeurs R, G, B = [160, 153, 141], on obtient respectivement -19.30893819 et 7.97372203. Une valeur négative implique la catégorie 0/ancienne et une valeur positive la catégorie 1/nouvelle. La prédiction de ces exemples est réussie.

* Test du modèle :

Pour valider ce modèle, nous avons donc fait 360 coupes où à chaque fois le couple de jeux entraînement-test est différent. Nous avons représenté la comparaison entre prédiction et vraie donnée à travers ce camembert. Les « true » sont toutes les fois où le modèle a prédit la bonne catégorie. Les « false » sont les fois où il s’est trompé.



* Résultat :

On a vu que le modèle pouvait se tromper. Nous pensons à deux explications opposées :

-quand le modèle ne reçoit pas les images qui caractérisent fortement leur catégorie (couleurs très prononcées) dans le jeu d’entraînement, elles manquent alors au modèle qui sera moins strict dans la catégorisation ;

-quand les images caractérisant faiblement leur catégorie (couleurs peu prononcées) passent en jeu de test, le modèle se trompe tout simplement.

## Vingt-sept modèles

<https://www.analyticsvidhya.com/blog/2021/05/lazy-predict-best-suitable-model-for-you/>

Nous n’allons pas nous intéresser à l’aspect scientifique du modèle utilisé alors nous allons essayer de boucler sur les données un maximum de types de modèles à la place de la régression logistique. Pour ça une solution pas mal du tout s’offre à nous : le package lazypredict. Il propose par exemple, d’appliquer en quelques lignes 27 algorithmes de classification sur des données.

* Première application :

Nous avons déjà appliqué le "modèle" (qui permet en fait de boucler sur 27 modèles) sur les données. Il propose ensuite un tableau avec en lignes les noms de classifieur et en colonnes des indicateurs de qualité ('Accuracy', 'Balanced Accuracy', 'ROC AUC', 'F1 Score'). Il utilise donc tous ces algorithmes *sklearn*, nous n’avons pas vérifié par algorithme s’il y avait des mauvais traitements des données et avons fait confiance au package. Le tableau et ses indicateurs changeaient à chaque lancement et donc on ne pouvait pas déterminer un meilleur algorithme ni même une liste.

* Deuxième application :

Nous avons appliqué une nouvelle fois la méthode *LeaveTwoDiffOut*. Mais les résultats des indicateurs étaient logiquement extrêmes : précisément 1, 0.5, 1/3 et 0. Néanmoins, nous avons repéré que 15 modèles avaient eu au moins une fois des indicateurs parfaitement performant, et 12 autres jamais.

# *Deep Learning*

# Réflexion et pistes d’amélioration

# Conclusion

# Bibliographie