

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 & Problématique 2](#_Toc119507638)

[1. Présentation des données 4](#_Toc119507639)

[2. 1ère Solution : Régression Logistique 5](#_Toc119507640)

[3. 2ème Solution : Data Augmentation + CNN 6](#_Toc119507641)

[4. Réflexion et pistes d’amélioration 6](#_Toc119507642)

[5. Conclusion 6](#_Toc119507643)

# Introduction & Problématique

Automatiser et industrialiser les processus de tâches répétitives telles que la reconnaissance automatique de documents est l’un des enjeux majeurs pour beaucoup d’entreprises. Cela permet à la fois de gagner du temps et gagner en efficacité tout en évitant les tâches répétitives. Ce qui peut également réduire les coûts en passant par une réduction de la masse salariale de l’entreprise.

Parmi ces documents qu’il peut être utile de classer de manière automatique, 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 - Anciennes cartes de séjour

Figure - Nouvelles cartes de séjour

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

Nous allons donc chercher à automatiser le processus de différenciation de ces deux 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 du *machine learning* seront travaillées :

-traitement de données ;

-*cross-validation* ;

-classification supervisée ;

-réseaux de neurones ;

-*data augmentation*.

Dans ce rapport nous détaillerons donc notre démarche, les problèmes rencontrés ainsi que les solutions trouvées à ces problèmes. Nous présenterons donc dans un premier temps les données et la façon dont nous les avons récoltées. Nous présenterons ensuite les différents modèles testés. Pour finir, nous présenterons une réflexion sur les résultats obtenus et les potentielles pistes d’améliorations.

# 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 Image. 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.



Figure - Nouvelles cartes de séjour

Figure - Anciennes cartes de séjour

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*.

Pour réaliser de la détection de type de cartes de séjour en utilisant du *machine learning* sous *Python*, nous avons pensé à plusieurs solutions. Dans un premier temps, nous avons voulu nous concentrer sur les colorations RGB des images (Rouge – Vert – Bleu) en constituant un modèle via la régression logistique.

Chaque image a un certain nombre de pixels. Chaque pixel a une valeur de rouge, une valeur de vert et une valeur de bleu. Pour chaque image on fait donc 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 avec pour chaque image sa catégorie (0/1, ancien/nouveau type) et trois valeurs numériques décrivant sa coloration.

# Proposition introductive : régression logistique

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

* Séparation du jeu d’entraînement du jeu de test :

Pour savoir comment valider le modèle, il va falloir garder des données de côté pour les tester sur un modèle entraîné : une méthode de *cross-validation*. Parmi les méthodes de *cross-validation* existantes, nous nous sommes intéressés au méthodes *LeaveOneOut* et *LeavePOut*. 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 modèle correspondant qu’il y a de données. Cette méthode est idéale et reste visualisable quand l’échantillon est très petit. 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 pour chaque jeu de test. Nous voulions néanmoins avoir une donnée par catégorie différente. Il n’existe pas de méthode toute faite par *sklearn*. Nous avons décidé de baptiser cette méthode "*LeaveTwoDiffOut*" (*leave two different data out*).

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeavePOut.html>

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

* 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 :

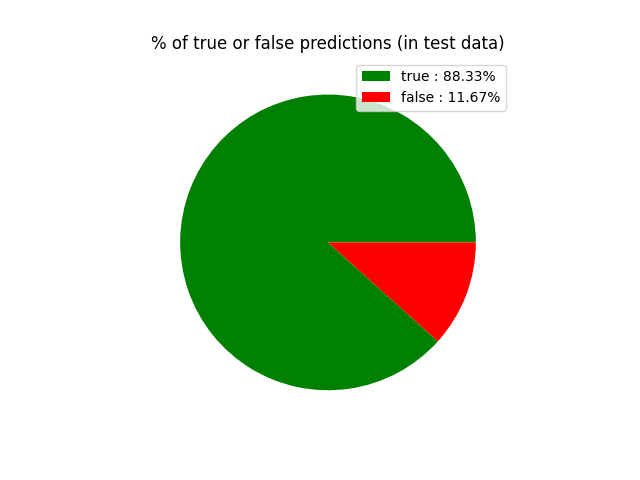
-0.74174263-0.39915359\*R + 1.16623781\*G +-0.75074005\*B

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.

* 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 résultat 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 ;

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

# Deuxième proposition : 27 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 précisément extrêmes : 1, 0.5, 1/3, 0. Néanmoins, nous avons repéré que 15 modèles avaient eu au moins une fois des indicateurs parfaitement performant, et 12 autres jamais.

# 2ème Solution : Data Augmentation + CNN

# Réflexion et pistes d’amélioration

# Conclusion