Haman Meryeme
Pathammavong Emilie

Airiau Stéphane

Projet de machine learning : Le Rubik’s cube

Dauphine | PSL
UNIVERSITÉ PARIS

Machine learning

2021/2022

I. Introduction

Ce projet a été réalisé dans le cadre de la première année de Master MIAGE à Paris Dauphine.
Notre travail consiste à trouver le coût optimal de la résolution d'un Rubik's cube. Pour mener ce projet, nous avons utilisé des algorithmes permettant de trouver le coût optimal correspondant à toutes les données de notre dataset.
L'objectif de notre travail est d' améliorer l'apprentissage des algorithmes avec des méthodes de machine learning.
Nous développerons tout d'abord les différentes parties de la réalisation du projet. Nous expliquerons ensuite la solution retenue. Finalement, nous conclurons en envisageant les ouvertures de ce projet, tout en questionnant sa précision.

II. Algorithmes testés qui ont été vus en cours

1. K plus proches voisins, arbres de décisions, classification naïve bayésienne

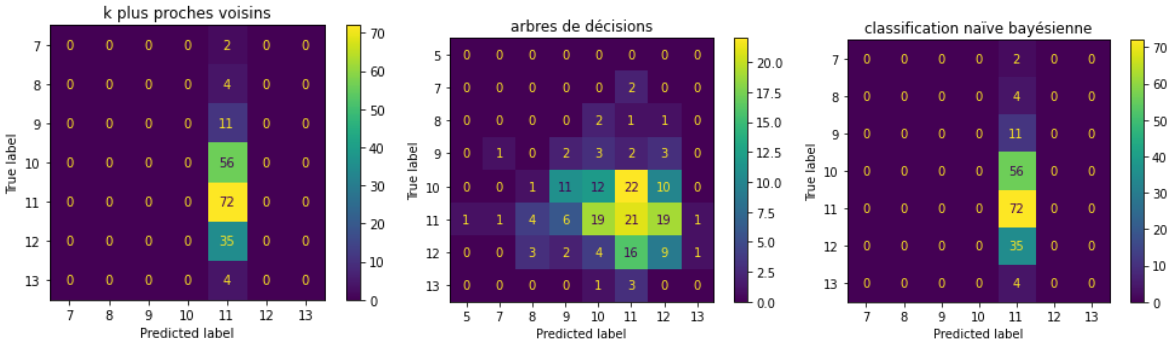
En premier, nous avons commencé par la vérification de X et y : nous avons vérifié que ces deux matrices ne contenaient aucun doublon, et qu'il n'y avait pas de redondance dans les données. Ensuite, nous avons partagé nos données d'entraînement en deux parties, une pour l'apprentissage (X_train et y_train) et une autre pour le test (X_test et y_test).

Pour l'algorithme des k plus proches voisins, nous avons testé avec plusieurs valeurs de k pour que le modèle soit le moins sensible aux valeurs aberrantes sans pour autant empiéter sur les autres classes. Nous sommes arrivés à une valeur de 75. Concernant le choix de la distance, après en avoir testé plusieurs, nous nous sommes arrêtés sur la distance de Manhattan, qui nous a paru la plus pertinente.

Pour l'arbre de décision, tenant compte de l'importance du choix des attributs dans la construction de l'arbre dans les prédictions, nous avons essayé les deux méthodes vues en cours pour mesurer la pureté des attributs, et sommes arrivé à la conclusion que la mesure d'entropie nous permettait d'obtenir un meilleur résultat.

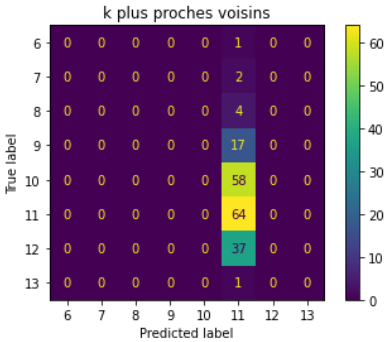
Par la suite, nous nous sommes contentées des trois algorithmes : Knn, Bayes et MLP, les résultats obtenus par l'arbre de décision ne nous paraissaient pas intéressants par rapport aux autres résultats.

Les 3 algorithmes ont donné le même score accuracy pour notre Xtest, mais les scores sur les données d'entraînement étaient différents. La matrice de confusion de l'arbre de décisions contenait des prédictions des classes 7, 8, 9, 10, 11, 12 et 13, contrairement aux deux autres qui ne prédisaient que des 11. Cependant nous pouvons douter de ce taux car il dépend des proportions des différentes classes, qui sont ici déséquilibrées.



2. Réseaux de neurones artificiels

Le score d'entraînement obtenu grâce aux réseaux de neurones a été meilleur, sauf qu'il ne prédisait que des 11, comme nous pouvons le voir sur l'image ci-dessous. Le score obtenu sur l'ensemble de test n'a pas changé par rapport aux autres algorithmes.



Conclusion :

Nous avons obtenu le même score en utilisant les quatres algorithmes vus en cours sur l'ensemble de test (Xtest), mais des scores d'entraînement différents. Ceci s'explique par la façon dont les données d'entraînement et de test ont été partagées, l'ensemble d'entraînement nous a uniquement permis de bien adapter les paramètres des algorithmes (K, poids, couches cachées, etc.), sans pour autant améliorer leurs performances.

III. Algorithmes testé qui n'ont pas été vus en cours :

1. Linear Discriminant Analysis :

Un algorithme qui fait partie des techniques d'analyse discriminante prédictive. Il s'agit d'expliquer et de prédire l'appartenance d'un individu à une classe (groupe) prédéfinie à partir de ses caractéristiques mesurées à l'aide de variables prédictives.

2. Voting Classifier :

Méthode de classification faisant partie des méthodes ensemblistes. Cela signifie qu'elles utilisent l'agrégation de multiples classificateurs "faibles" pour arriver à un meilleur modèle et ainsi faire leurs prédictions. Concernant le voting classifier, il fait ses prédictions en utilisant les prédictions des différents classificateurs et renvoie le résultat le plus fréquent.

a. Bayes / MLP Classifier :

Nous avons tout d'abord essayé en agrégeant les modèles de Bayes et MLP classifier.
résultat : Le score de MLP classifier

b. AdaBoostClassifier / RandomForestClassifier :

Nous avons ensuite testé avec les modèles AdaBoost et RandomForest. L'AdaBoostClassifier est aussi une méthode ensembliste. Sa particularité réside dans le fait qu'elle attribue des poids aux classificateurs : un classificateur qui a un moindre taux d'erreur aura plus de poids qu'un classificateur qui fait plus d'erreurs.
La méthode RandomForestClassifier tient sur le fait que plus nous avons d'arbre de décisions, meilleurs seront les résultats. Elle crée alors des arbres de décisions sur un ensemble de données choisies aléatoirement, récupère la prédiction de chaque arbre et sélectionne la meilleure solution d'après un vote.
résultats : Nous avons exécuté le code plusieurs mais nous n'avons jamais eu de résultat.

3. XGBoost :

Cette méthode fait aussi partie des méthodes ensemblistes. Sa particularité est que les classificateurs qu'elles utilisent sont des arbres décisionnels et ceux-ci sont élagués jusqu'à ce qu'ils soient assez performants, quitte à être complètement supprimés si besoin. La méthode pour optimiser est la méthode de descente de gradient, que nous avons vu en cours.

4. Support Vector Classifier (SVC) :

Méthode faisant partie des Support Vector Machine, qui est un algorithme utilisant la composante vectorielle des éléments du jeu de données d'apprentissage afin d'en déterminer une orientation préférentielle, il sera donc possible de définir une portée par cette orientation donnée par la composante vectorielle.
Nous n'avons malheureusement pu obtenir de résultat, mais après recherche nous pouvons dire que nous n'aurions probablement pas obtenu un résultat plus intéressant, ce type d'algorithme étant intéressant dans les cas où le nombre de dimensions est plus grand que la taille de l'échantillon, ce qui n'est pas notre cas.

5. Conclusion :

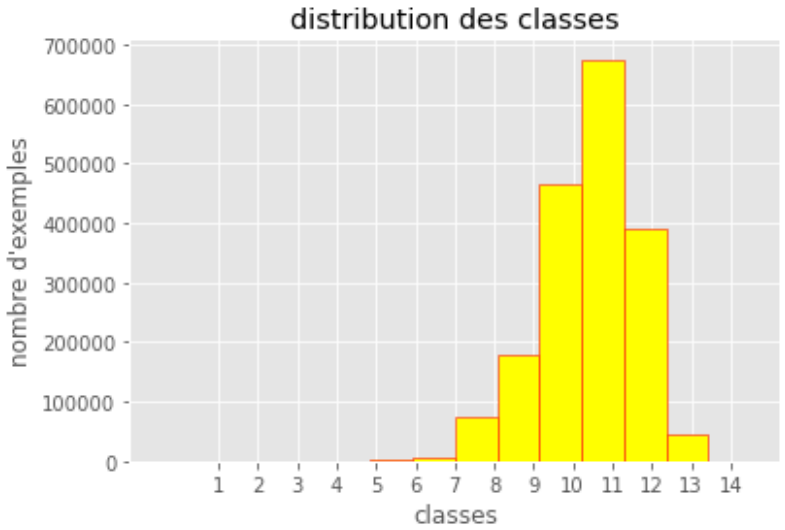
Nous avons utilisé d'autres algorithmes en pensant que cela pouvait améliorer les prédictions de nos données, et donc améliorer notre score. Certains parmi ces algorithmes nous ont permis effectivement d'améliorer le score sur l'ensemble d'entraînement, mais le score sur l'ensemble des tests Xtest quant à lui n'a pas changé.
L'utilisation de ces algorithmes nous a permis de connaître les différentes manières que les modèles utilisent pour découvrir les patterns enfouies dans les données.

Conclusion :

L'entraînement de ces différents algorithmes sur notre jeu d'entraînement et les prédictions obtenues nous ont permis de comprendre que, afin d'améliorer la prédiction, il ne suffisait pas uniquement d'utiliser plusieurs algorithmes, mais il fallait surtout traiter les données, détecter la problématique et ensuite trouver des solutions.

IV. Traitement des données :

1. La distributions des classes par rapport aux observations :



Comme nous pouvons le constater, les données sont complètement déséquilibrées, les classes 11, 10, et 12 sont majoritaires, puis nous avons les classes 8,9 et 13 et enfin les classes 1, 2, 3, 4, 5, 6, 7 et 14 qui représentent une minorité dans notre échantillon.

2. Identifications des patterns :

Pour identifier des patterns, nous avons créé des matrices **Xi** et **yi** correspondantes à chaque classe dans notre **y_unique.** Les patterns identifiés :

a. Les positions Pos6, Pos14 et Pos22 sont communes entre toutes les observations.
b. Plusieurs observations ont les positions Pos5, Pos15 et Pos18 égales respectivement à 1, 2 et 3.

Suite à ces observations, nous avons défini un ensemble de données composé uniquement des observations vérifiant notre deuxième pattern, et sur lequel nous avons entraîné nos algorithmes.

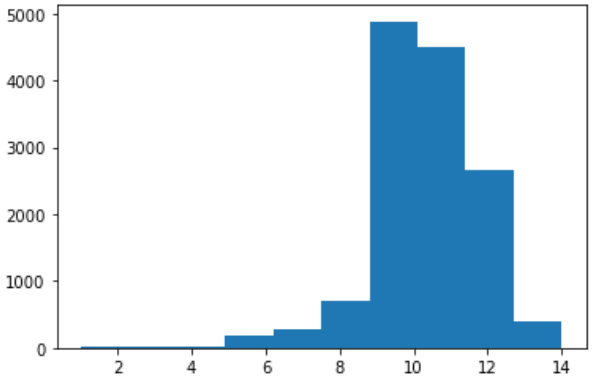


Figure 6 : La distribution des classes au sein de la matrice X_5_15_18

Les scores obtenus sur l'ensemble de tests en entraînant les algorithmes de **Knn** et **Bayes** sur cet ensemble de données avant et après la suppression des colonnes en commun (Pos5, Pos6,

Pos14, Pos15, Pos18 et Pos22) ne se sont pas améliorés, contrairement à celui de **MLP** qui a pu prédire également des 12 en plus des 11.
Cette amélioration est dûe au fait qu'il y'a moins d'observations, et vu que le score est calculé à partir du nombre des exemples **n**, le score augmente en diminuant n.

	precision	recall	f1-score	support
6	0.00	0.00	0.00	2
7	0.00	0.00	0.00	3
8	0.00	0.00	0.00	10
9	0.00	0.00	0.00	22
10	0.00	0.00	0.00	32
11	0.31	1.00	0.47	42
12	1.00	0.04	0.08	24
13	0.00	0.00	0.00	2
accuracy			0.31	137
macro avg	0.16	0.13	0.07	137
weighted avg	0.27	0.31	0.16	137

Figure 7 : Le rapport de la prédiction de MLP sur la matrice X_5_15_18

3. Traitement des observations dont la classe est 10, 11 ou 12 :

Nous avons ensuite traité les observations dont les classes sont majoritaires en concaténant les matrices X_i et y_i avec i appartenant à {10, 11, 12}

	precision	recall	f1-score	support
10	0.50	0.00	0.00	4675
11	0.45	1.00	0.62	6848
12	0.00	0.00	0.00	3797
accuracy			0.45	15320
macro avg	0.32	0.33	0.21	15320
weighted avg	0.35	0.45	0.28	15320

Figure 8 : Le rapport de la prédiction de MLP, KNN, Bayes sur la matrice X_10_11_12

	precision	recall	f1-score	support
10	0.34	1.00	0.50	3960
11	0.00	0.00	0.00	3946
12	0.11	0.00	0.00	3833
accuracy			0.34	11739
macro avg	0.15	0.33	0.17	11739
weighted avg	0.15	0.34	0.17	11739

Figure 9 : Le rapport de la prédiction de MLP sur la matrice X_10_11_12 avec une répartition équitable

```
Train score : 1.000000
Test score : 0.337337
[[3960    0    0]
 [3946    0    0]
 [3833    0    0]]
              precision    recall  f1-score   support

          10       0.34      1.00      0.50      3960
          11       0.00      0.00      0.00      3946
          12       0.00      0.00      0.00      3833

    accuracy                           0.34     11739
   macro avg       0.11      0.33      0.17     11739
weighted avg       0.11      0.34      0.17     11739
```

Figure 10 : Le rapport de la prédiction de MLP sur les données restantes de la matrice X_10_11_12

Dans les trois situations, les scores d'entraînement se sont améliorés grâce à la réductions de **n**, le nombre d'exemples de test, mais le score de notre algorithme MLP sur l'ensemble des données de **Xtest** ne s'est pas amélioré, parce que nous l'avons entraîné sur un ensemble de données qui est déjà majoritaire.

4. Traitement des observations dont la classe est 8, 9 ou 13 :

Puis, nous avons traité les observations dont les classes sont majoritaires en concaténant les matrices X_i et y_i avec i appartenant à {8, 9, 13}

	precision	recall	f1-score	support
8	0.00	0.00	0.00	603
9	0.62	1.00	0.77	1765
13	0.00	0.00	0.00	457
accuracy			0.62	2825
macro avg	0.21	0.33	0.26	2825
weighted avg	0.39	0.62	0.48	2825

Figure 11 : Le rapport de la prédiction de MLP sur les données de la matrice X_8_9_13

Le score d'entraînement s'est amélioré à 0.62 grâce à la distribution des observations selon les classes, mais comme nous pouvons le constater sur la Figure 11, le modèle n'a prédit correctement que les observations dont la classe est majoritaire. (cf Figure 11)

5. Traitement des observations dont la classe est 1, 2 3, 4, 5, 6, 7 ou 14 :

Enfin, nous avons traité les observations dont les classes sont majoritaires en concaténant les matrices X_i et y_i avec i appartenant à {1, 2, 3, 4, 5, 6, 7, 14}

	precision	recall	f1-score	support
4	0.00	0.00	0.00	6
5	0.00	0.00	0.00	9
6	0.00	0.00	0.00	38
7	0.76	1.00	0.87	173
14	0.00	0.00	0.00	1
accuracy			0.76	227
macro avg	0.15	0.20	0.17	227
weighted avg	0.58	0.76	0.66	227

Figure 12 : Le rapport de la prédiction de MLP sur les données de la matrice X_1_2_3_4_5_6_7_14

Le score d'entraînement s'est encore amélioré à 0.76 grâce à la distribution des observations selon les classes, mais l'algorithme ne prédit toujours pas les observations dont la classe est minoritaire dans l'ensemble de tests. (cf Figure 12)

Conclusion :
Le traitement des différentes classes et observations nous ont permis de comprendre que la problématique majeure de notre projet est : **le déséquilibre au sein de notre échantillon y,** les classes de 1 à 14 ne sont pas représentées de façon égale.

6. Identifications d'autres patterns : Clusters

Afin de trouver d'autres patterns, nous avons utilisé les clusters. Nous avons essayé d'analyser les différentes sous-classes générées pour identifier les caractéristiques qui les identifiaient, mais cette analyse ne nous a permis de définir aucun autre pattern.

V. Améliorations des données

Afin de contourner le déséquilibre des classes, nous avons utilisé une méthode de data-level qui consiste en une transformation opérée sur nos données d'entraînement, et d'autres qui reposent sur des modifications des modèles utilisés pour qu'ils soient plus adaptés à notre problème.

1. Définition des faces et diminution des colonnes

Pour entraîner notre algorithme sur moins de colonnes, nous avons essayé de déterminer la structure du Rubik's cube, en définissant les faces, et au lieu d'avoir 24 colonnes qui représentaient les 24 positions, nous avons obtenu 6 listes, chacune représentant une face et une couleur différentes.

Cette méthode n'a pas permis d'améliorer l'apprentissage de notre algorithme, et le score obtenu sur l'ensemble de tests était plus bas que tous les scores que nous avions déjà obtenus à cause de la surabondance des 11.

2. Équilibrer les poids des classes

a. Random Forest Classifier

Nous avons ensuite voulu utiliser l'algorithme **RandomForestClassifier** avec le paramètre **class_weight="balanced",** afin d'ajuster les poids de classes en fonction du nombre d'échantillon de chacune, nous n'avons malheureusement pu obtenir de résultat à cause de la mémoire, mais après avoir cherché, cet algorithme n'aurait peut être pas apporter suffisamment d'améliorations.

a. Class Weights et Logistic Regression

La régression logistique est un modèle de régression binomiale, il s'agit donc de modéliser au mieux un modèle mathématique simple à des observations réelles nombreuses, i.e associer à un vecteur de variable aléatoires une variable aléatoire binomiale. Nous obtenons une fonction qui définit la régression logistique et le problème apparaît alors comme un problème d'optimisation où, à partir des données, nous essayons d'obtenir le jeu de paramètre qui permet à la courbe de notre fonction de coller au mieux à nos données.
Grâce au modèle **utils** de sklearn, nous avons calculé les poids de classes (**class_weights),** que nous avons passés en paramètre de l'algorithme LogisticRegression().

Cette méthode n'a pas permis d'améliorer les données, ni d'obtenir une meilleure prédiction, (Ceci est peut être dû au calcul des poids).

3. Deep Learning

a. Random Under Sampler

Consiste à supprimer aléatoirement des exemples des classes majoritaires et ainsi les enlever de l'ensemble d'apprentissage. N'est seulement que si les classes minoritaires possèdent suffisamment de données. A noter que comme les exemples sont supprimés aléatoirement, il est impossible de garder les "bons" exemples.
Cette méthode n'a pas été pertinente, car l'algorithme a supprimé aléatoirement les observations des classes majoritaires, et donc il apprenait uniquement sur un faible pourcentage des autres observations, par conséquent il ne s'entraînait pas suffisamment pour prédire les bonnes distances.

	precision	recall	f1-score	support
3	0.00	0.00	0.00	0.0
9	0.00	0.00	0.00	1.0
accuracy			0.00	1.0
macro avg	0.00	0.00	0.00	1.0
weighted avg	0.00	0.00	0.00	1.0

Figure 13 : Le rapport de la prédiction de RUS sur les données de la matrice X_sm et y_sm

4. One Hot Encoder

Le One Hot Encoder est un type de représentation vectorielle dans lequel tous les éléments d'un vecteur valent 0, sauf un, qui a pour valeur 1, où 1 représente un booléen spécifiant une catégorie de l'élément.
Cette méthode a permis d'avoir une représentation vectorielle de nos données, mais la façon dont elles sont réparties n'a pas changé. Le score d'entraînement s'est amélioré mais celui sur l'ensemble des tests n'a pas augmenté.

5. Preprocessing

a. Min Max scaler

Normalise les données pour qu'elles correspondent à un intervalle donné.

b. Robust Scaler

Méthode de normalisation des données utilisant des méthodes statistiques qui lui permet d'être robuste face aux valeurs aberrantes. Pour se faire, elle supprime la valeur médiane et met à l'échelle selon le quartile.

c. Normalization

Normalise les données à l'unité indiquée.

d. Standardization

Standardise les données en enlevant la moyenne et divisant par l'écart-type.

Ces quatres méthodes n'ont pas permis l'équilibrage des données, et donc elles ne les ont pas améliorées. En effet, le prétraitement des données permet d'éliminer les données indésirables grâce au nettoyage des données, ce qui permet d'avoir un ensemble de données contenant des informations plus précieuses après l'étape de prétraitement pour la manipulation des données plus tard dans le processus d'exploration de données, sauf que dans notre cas toutes les données sont désirées.

VI. Solution retenue :

1. Pandas dummies for Encoding

Il s'agit d'une stratégie d'encodage permettant de convertir une caractéristique catégorielle en un format de vecteur numérique. La fonction pd.get_dummies () est une fonction de Pandas qui effectue un codage fictif en une seule ligne de code.
Cette méthode a permis à notre algorithme de prédire d'autres classes autres que la classe 11. (cf Figure 13)

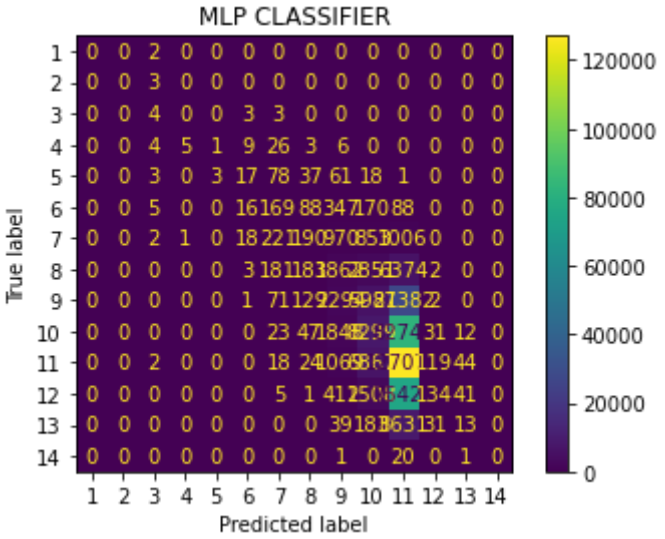


Figure 13

En changeant le nombre de couches cachées de nos réseaux de neurones, ainsi que le nombre des itérations, le score s'est amélioré :

En local :

0.364488 ⇒ 0.376279 ⇒ 0.379997 ⇒ 0.3769980.8583 ⇒ 0.380273

Sur le site :

0.853522982123805 ⇒ 0.80304178370022 ⇒ 0.790485988634137 ⇒ 0.790070111263527 ⇒ 0.782225052801184

VII. Conclusion :

Rubik's cube est notre premier projet en Machine Learning, il fût intéressant pour nous sur l'aspect de l'apprentissage automatique. Il nous a permis de comprendre les principes du Machine Learning et leurs applications en pratique. Pendant sa réalisation, nous avons rencontré plusieurs difficultés au niveau du temps et du matériel à notre disposition.

C'était vraiment un défi pour nous de tester tous les algorithmes sur notre dataset, et d'améliorer nos données, notamment en utilisant la technique SMOTE du deep learning, que nous avons voulue appliquer sur notre matrice pour l'équilibrer.

Ce projet nous a permis de développer nos compétences techniques en python et nous a également permis de comprendre en profondeur les notions vues en cours. Nous avons compris la différence entre les différents algorithmes, l'importance de la bonne simulation des paramètres des algorithmes, de la matrice de confusion et le rapport de classification, et surtout de ne pas se baser uniquement sur le score obtenu sur le jeu d'entraînement.

Ils nous a également permis de développer des compétences de nos savoir-être comme la patience lorsque nous attendions nos algorithmes de tourner pendant des heures, ainsi que la gestion du stress quand nos ordinateurs crashaient.

Malgré les difficultés rencontrées, nous avons comme objectif de continuer ce projet pour améliorer la distribution des classes en utilisant les méthodes de data-level ainsi que celles d'algorithm-level pour équilibrer les données, comprendre en détails les mécanismes de l'apprentissage et avoir un score de 0 .