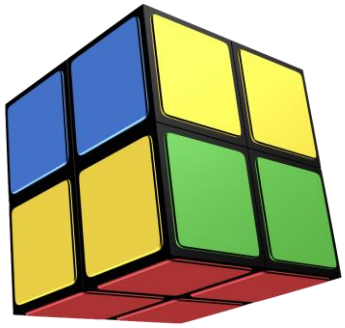
Dauphine | PSL
UNIVERSITÉ PARIS

Machine Learning & Application

Projet Rubik's Cube

Anna Grech
Martin Le Digabel



I - Étude des données

Tout d'abord nous avons réalisé une étude des données qui nous étaient fournies. Notre base de données train_input était composée de 25 colonnes (l'ID et les positions), et train_output de deux colonnes (l'ID et le nombre de coups minimal pour résoudre le Rubik's cube). La base de données n'a pas de valeur *null* et toutes les données sont de type Integer. Nous savons que la répartition de la donnée est très déséquilibrée : nous avions à disposition seulement 3 Rubik's cubes solvables en un coup et plus de 600 000 solvables en 11 coups.

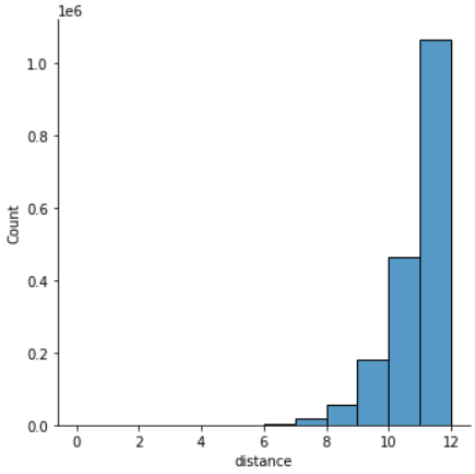


Fig. 1 : Répartition des valeurs de la classe y.

Nous avons réalisé la matrice de corrélation entre les différentes positions, ce qui nous a montré que les corrélations étaient très faibles entres les positions, voire parfois nulles, entre les pos 6, 14 et 22. Ces colonnes ont toujours la même valeur peu importe le Rubik's Cube

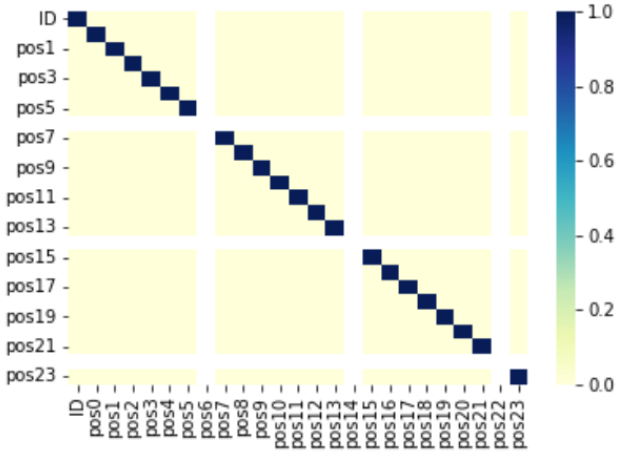


Fig. 2 : Matrice de corrélation de notre DataFrame X_train

II – Re-découpage en train/test et Validation croisée

Ne pouvant pas accéder à la partie y_test de notre Dataset, nous avons décidé de tester par la suite nos modèles uniquement sur la partie train que nous avons recoupé en quatre parties : X_train, X_test, y_train et y_test.

Nous avons aussi décidé de valider nos modèles par la méthode de la validation croisée pour vérifier que le découpage train/test des données n'influençait pas la performance de nos modèles. Afin de faire cette validation croisée, nous avons décidé d'utiliser l'algorithme K-Stratified Fold qui nous a permis de garder les mêmes proportions des différentes classes dans la partie train et test. Cela était d'autant plus important du fait que notre Dataset est très déséquilibré.

III – Deux approches différentes

a - Vue comme un problème de régression – Régression linéaire et ElasticNet

Dans un premier cas, nous avons souhaité modéliser l'exercice comme un problème de régression. Le but est alors d'obtenir un résultat entier situé entre 1 et 14. Sachant que la matrice de corrélation affichait des corrélations très faibles entre les variables, il nous semblait que cette approche aurait sûrement des résultats peu performants comparés des résultats obtenus via des algorithmes non-linéaires.

Nous avons donc entraîné nos données via les algorithmes Régression Linéaire et ElasticNet. Nous avons obtenu des résultats de MAE similaires pour les algorithmes de Régression Linéaire et d'ElasticNet. Cela peut aussi se justifier du fait que la pénalisation sur les variables via ElasticNet ne fonctionne pas bien dû au fait qu'il n'existe pas vraiment de coefficients plus significatifs que d'autres dans notre dataset (chaque position semble avoir la même importance dans la prédiction du nombre de coup). Cela était déjà visible via notre matrice de corrélation.

	train score	test score
0	-0.927998	-0.928025
1	-0.928004	-0.928029
2	-0.928010	-0.928014

	train score	test score
0	-0.927998	-0.928025
1	-0.928004	-0.928029
2	-0.928010	-0.928014

Fig. 3 : Tableau des scores de MAE de LinearRegression et ElasticNet (avec validation croisée)

b - Vue comme problème de classification

Dans un second cas, nous avons souhaité modéliser l'exercice comme un problème de classification. Nous avons souhaité utiliser plusieurs algorithmes pour entraîner nos données : K-Nearest Neighbors, un réseau de neurones MLPClassifier ainsi qu'Adaboost.

Nous avons retenu que l'algorithme donnant le meilleur résultat était le réseau de neurones avec une erreur moyenne absolue de 0,85 environ. L'algorithme Adaboost offrait un résultat presque similaire sur nos données d'entraînement et les K-Nearest Neighbors n'ont finalement pas été retenus car l'algorithme est très lent dû à la très grande taille de notre base de données.

	train score	test score
0	-0.858316	-0.858310
1	-0.858334	-0.858304
2	-0.858318	-0.858307

Fig. 4 : Tableau des scores de MAE du réseau de neurones MLP (avec validation croisée)

IV – Modifications de la base de données

Nous retenons que l'algorithme délivrant les meilleurs résultats est le réseau de neurones. Par la suite, nous essaierons de retravailler notre base de données afin d'améliorer nos scores.

a. Analyse des positions

Dans un premier temps, nous nous sommes intéressés à mieux comprendre notre base de données.

Nous avons essayé de comprendre comment étaient réparties les pos[i] par face dans l'espace. Les positions ne nous semblaient pas ordonnées.

Pour cela nous avons extrait la liste des pos[i] des Rubiks cube qui étaient résolvables en un coup. Un Rubik's cube solvable en un coup est un Rubik's cube dont deux faces opposées sont déjà résolues et les autres faces ne peuvent contenir que deux couleurs simultanément.

Avec l'études de ces trois séries, nous avons trouvé une solution viable avec pour faces opposées les faces :

- Face 1 opposée à Face 2
- Face 3 opposée à Face 4
- Face 5 opposée à Face 6

(Voir *Annexe : Réorganisation des positions sous Excel et Simulation 3D*)

Nous n'avons cependant pas réussi à déterminer l'ordre de ces facettes sur chaque face. Cela était cohérent sur les Rubik's cube à résoudre en 1 coup. Mais dès que cette réorganisation

était appliquée à une série d'un Rubik's cube résolvable en deux coups, l'ordre des facettes sur chaque face ne correspondait pas, bien que les couleurs étaient sur les bonnes faces.

Nous nous sommes donc basés sur l'hypothèse que nous pouvions connaitre les couleurs présentes sur chaque face mais pas l'ordonnancement des couleurs au sein d'une face.

Nous avons donc créé une nouvelle base de données afin d'essayer d'améliorer notre modèle : **dmodif**, copie de notre base de données initiale (X_train et X_test fusionnés).

b. Création des colonnes 'nb_couleurs_face[i]'

Dans un premier temps, nous avons créé une série de 6 colonnes 'nb_couleurs_face[i]' qui représentent le nombre de couleurs différentes au sein d'une face. Exemple : Si la face1 = [1, 1, 1, 2], le nombre de couleurs différentes présentes sur la face1 est 2. (Voir ci-dessous Fig. 5)

c. Création des colonnes face[i]_color[j]

Dans un second temps, nous avons créé une série de 36 colonnes 'face[i]_color[j]' qui représentent chacune le nombre de fois où la couleur j apparaît sur la face i. (Voir ci-dessous Fig. 5)

	pos0	pos1	pos4	pos5	nb_couleurs_face1	face1_color1	face1_color2	face1_color3	face1_color4	face1_color5	face1_color6
0	4	1	6	2	4	1	1	0	1	0	1
1	6	5	2	2	3	0	2	0	0	1	1
2	5	3	3	1	3	1	0	2	0	1	0
3	5	5	2	1	3	1	1	0	0	2	0

Fig. 5 : Exemple de la face 1 représentée par les colonnes ajoutées 'face[i]_color[j]' et nb_couleurs_face[i]

d. Suppression des colonnes dupliquées

Nous avons supprimé les lignes en double de notre dataset (via la fonction .drop_duplicate).

V- Entraînement du réseau de neurones sur 2 bases de données modifiées

Nous avons créé une base de données **dmodif2** ne contenant que les colonnes créées sur l'hypothèse que nous avions les couleurs par face de notre Rubik's Cube.

Nous avons ensuite entraîné un réseau de neurones une fois avec **dmodif** et **dmodif2**.

Les résultats sont équivalents pour les deux bases de données et similaires avec la base de données initiale.

	train score	test score
0	0.858293	0.858344

	train score	test score
0	0.858233	0.858337

Fig. 6 : Tableau des scores de MAE du réseau de neurones MLP sur dmodif et dmodif2

VI- Tuning des hyperparamètres de notre modèle MLP

Finalement, nous avons tenté d'améliorer notre modèle en cherchant les meilleurs paramètres possibles via l'algorithme GridSearchCV.

Les paramètres ainsi trouvés sont :

{'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'solver': 'adam'}

Ceux-ci nous ont apporté un score de : 0.858906525573192

Conclusion

Lors de cet exercice, nous nous sommes attachés à visualiser et analyser nos données.
Nous avons ensuite modélisé le problème via une vue de régression puis de classification.
Nous retenons que modéliser le problème comme un problème de classification via un algorithme de réseau de neurones nous paraissait cohérent.
Nous avons donc entraîné une première fois nos données initiales avec un algorithme MLPClassifier avec ses hyperparamètres par défaut.
Nous avons ensuite entraîné une seconde fois nos données modifiées, comme expliqué précédemment, avec le MLPClassifier dont les hyperparamètres étaient sélectionnés.

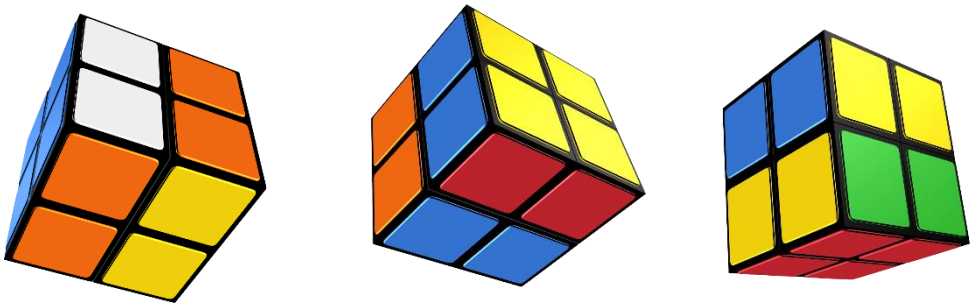
Nos résultats dans les deux cas sont assez similaires et finalement nos modifications apportées aux données ont été assez peu concluantes. Si nous avions pu comprendre plus en profondeur le placement des données au sein d'une face, nous aurions pu continuer d'améliorer notre base de données.

L'utilisation du réseau de neurones reste en tout cas pour nous la meilleure façon d'utiliser nos données.

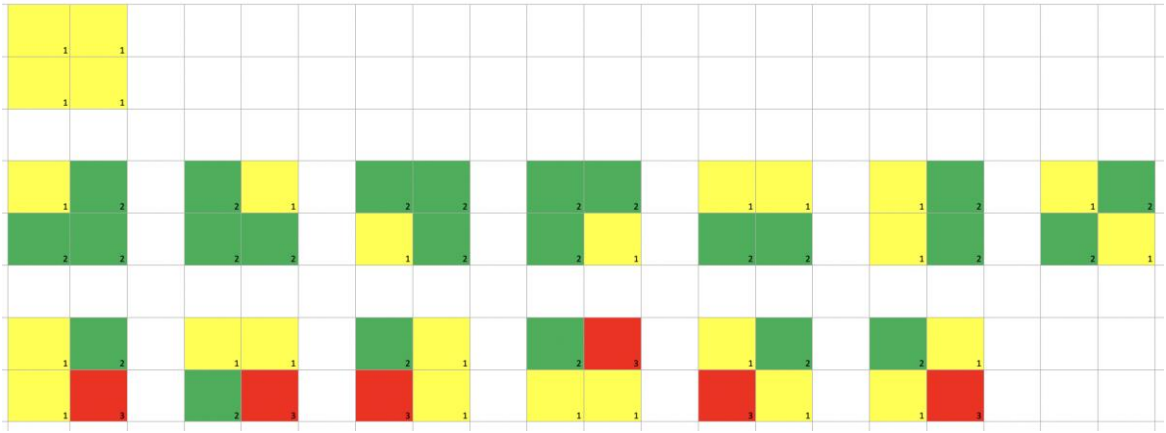
ANNEXE : *Réorganisation des positions sous Excel et Simulation 3D*

Série 1	5	1	6	3	5	1	6	3	2	4	4	2	2	4	4	2	1	3	3	1	6	5	5	6
Série 2	1	1	6	6	1	1	6	6	3	5	4	2	3	5	4	2	4	4	3	3	2	2	5	5
Série 3	2	2	4	4	1	1	6	6	6	1	1	6	2	4	4	2	3	3	3	3	5	5	5	5
		Face				Arrière				Droite				Gauche				Bas				Haut		
Série 1 par face	5	1	5	1	3	6	3	6	2	2	2	2	4	4	4	4	1	3	1	3	6	5	6	5
Série 2 par face	1	1	1	1	6	6	6	6	2	3	2	3	5	4	5	4	3	3	4	4	2	2	5	5
Série 3 par face	1	1	2	2	4	4	6	6	2	2	6	6	4	4	1	1	3	3	3	3	5	5	5	5

Extraction des séries de position des cubes solvables en 1 coup et réorganisation par face



Simulation 3D et représentation dans l'espace pour aide à la compréhension



Représentation des différents schémas de couleurs possibles pour implémentation des nouvelles colonnes dans la base de données