Maison du Monde - Test Technique

Cécile Hannotte

1 Installation du projet

Pour installer les dépendances du code et de l'environnement, possiblement installer les dépendances du requierements.txt pour conda sous windows ou du pip_requirements.txt pour pip sur Windows et Linux. La première partie a été développé en mode recherche sur Jupyter Notebook avant d'être installée dans un serveur flask pour l'API. Le code est disponible sur github.

2 Machine Learning

Cette partie regroupe l'analyse de données et la prédiction de la catégorie d'un objet. Elle reprends les démarches de l'analyse en détails. L'objectif de la partie ML est de prédire la catégorie d'un objet entre trois classes : 'mlp', 'deco'

et 'meuble' en fournissant ses dimensions et son poids. Nous sommes donc face à un problème de classification à 3 classes.

2.1 Analyse du dataset et nettoyage

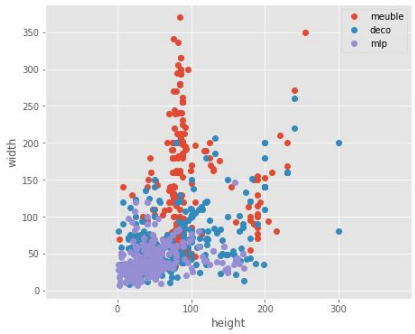
Nous remarquons que nous avons 900 données pour 5 variables. L'étiquette à prédire se trouve sur la dernière colonne, la colonne 'activity'. Les autres variables seront les données d'entrées pour la prédiction, elle sont au nombre de quatre : La hauteur, la largeur, la profondeur et le poids de l'objet à prédire.

```
[2]  1 mdm_data = pd.read_csv('dataset.csv')
     2 mdm_data.head()
```

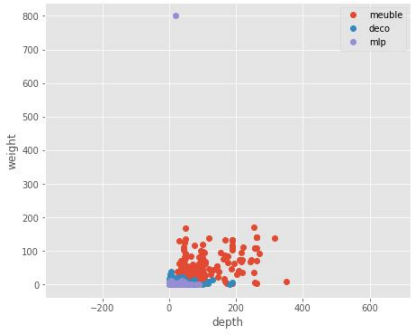
	height	width	depth	weight	activity
0	90.5	42.0	2.0	1.25	mlp
1	60.0	50.0	2.0	1.20	mlp
2	80.0	59.6	2.0	7.40	mlp
3	46.0	46.0	2.0	1.45	mlp
4	93.0	21.0	2.0	1.30	mlp

FIGURE 1 – Début du fichier csv du dataset

Nous remarquons que le fichier **est trié** par type d'objet. Par conséquent nous devons impérativement mélanger les données pour l'entraînement afin qu'il ne s'adapte pas au tri effectué. L'étiquette étant une chaîne de caractères, elle sera binarisée pour éviter au réseau une plus grande complexité. Le reste des données étant des flottants, nous n'apporterons pas de modification dans les valeurs.



(a) Largeur d'un objet en fonction de sa hauteur



(b) Profondeur d'un objet en fonction de son poids

FIGURE 2 – Premières analyses du dataset : Nuage de points

En traçant un premier nuage de point de nos données pour voir leur repartissions, nous remarquons que certaines données ont des valeurs très éloignées de la moyenne des autres données. Il existe donc des valeurs aberrantes. Notons aussi que les catégories 'mlp' et 'deco' se confondent et se superposent dans les deux graphiques, mais que la catégorie 'meuble' quant à elle se distingue assez bien. Il ne semble donc pas y avoir d'erreur dans les données, car de façon logique, un meuble a des dimensions significativement plus grande qu'un objet de déco ou mlp, de par sa nature. Il est donc aussi plus lourd. La distinction entre 'mlp' et 'deco' semble ici un peu plus complexe. Nous émettons l'hypothèse que les erreurs du futur réseau de prédiction, viendront du fait de son hésitation entre ces deux catégories.

Ce que nous retenons surtout de ces figures, est le fait qu'il existe des valeurs hors-normes dans le graphique et que nous sommes mieux de les supprimer par la méthode du score Z.

En statistique, une valeur aberrante est un point d'observation qui est éloigné des autres observations. Le Z-score consiste à trouver la distribution des données où la moyenne est 0 et l'écart-type 1, c'est-à-dire la distribution normale. En calculant le score Z, nous redimensionnons et centrons les données et recherchons les points de données qui sont trop éloignés de zéro, c'est à dire les valeurs aberrantes. Dans la plupart des cas, un seuil de 3 ou -3 est utilisé, c'est-à-dire que si la valeur du score Z est supérieure ou inférieure à 3 ou -3 respectivement, ce point de données sera identifié comme aberrant car il représentera les 0.3% des valeurs de la distribution normale.

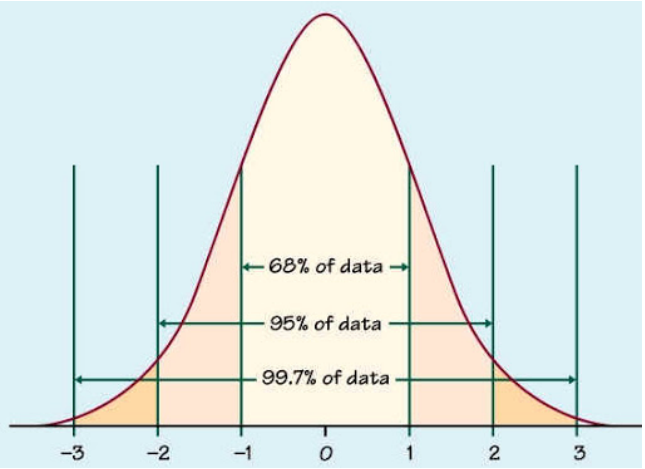


FIGURE 3 – Distribution normale et Z-score

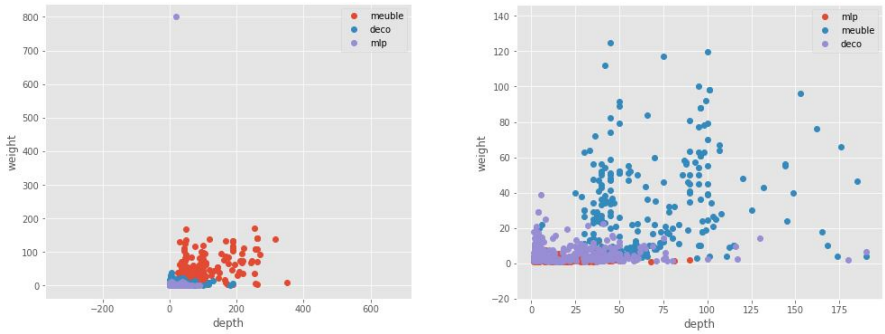
Comme nous avons peu de données, nous fixons un score élevé pour éviter de supprimer une grosse partie de nos données. Même en fixant un score de 3, nous remarquons que nous supprimons 52 données sur nos 900 initiales, c'est à dire 5.8% de nos données.

```
from scipy import stats
import numpy as np
z = np.abs(stats.zscore(mdm_data.iloc[:, :4]))
threshold = 3
print(np.where(z > threshold)[0])
row_outliers, _ = np.where(z > threshold)
new_mdm = mdm_data.drop(row_outliers)
new_mdm.shape
```

```
[135 303 304 307 310 311 315 316 319 336 370 399 611 645 658 704 709 710
 711 712 717 743 746 747 748 749 750 754 755 756 756 757 757 758 758 759
 779 802 849 849 853 854 854 855 855 856 857 858 858 858 859 859 860 860
 860 885 885 887 888 890 891 892 893 894 895 896 897 898 899 899 899]
(843, 5)
```

FIGURE 4 – Index retirés pour un Z-score de 3

En supprimant les valeurs aberrantes, nous remarquons que nous améliorons la représentation de nos données.



(a) Largeur d'un objet en fonction de sa hauteur, avec outliers

(b) Profondeur d'un objet en fonction de son poids sans outliers

FIGURE 5 – Comparaison du nuage de points avec et sans les valeurs aberrantes

2.2 Prédiction simple (Arbre de décision)

Actuellement, les noms des 3 catégories ne sont pas importants, nous les encodons donc comme suit : 001 pour mlp, 010 pour meuble et 100 pour déco.

Comme nous ne travaillons qu'avec des valeurs numériques, nous pouvons appliquer directement une normalisation à X pour voir s'il y a des valeurs aberrantes. Cette transformation doit s'opérer APRÈS l'affectation de Train/test pour éviter les fuites de données (data leakage). En effet si nous récoltons la moyenne et l'écart type de nos données sur tout le dataset, cela voudrait dire que nous prenons en compte celle du dataset de test, et donc que nous utilisons ces informations pour l'entraînement. La première étape est donc de diviser les données en un dataset d'entraînement et un dataset de test sans les normaliser. Ensuite, nous calculons la moyenne et l'écart type des données d'entraînement que nous sauvegardons. Nous normalisons nos données d'entraînement avec ses valeurs. Ensuite, nous reprenons ces mêmes valeurs pour normaliser le dataset

de test. Nous sauvegardons le scaler (les variables de standardisation de notre entraînement) pour les ajuster sur les futures données. Une prédiction classique s'effectue ainsi :

1. Récupérer les 4 variables (height, width, depth, weight)
2. Normaliser ces données avec la moyenne et l'écart-type sauvegardée précédemment
3. Charger le modèle de prédiction entraîné
4. Décoder la réponse et l'afficher

Nous commençons notre prédiction en utilisant un arbre de décision classique. Un arbre de décision est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Les différentes décisions possibles sont situées aux extrémités des branches, et sont atteintes en fonction de décisions prises à chaque étape. Dans notre cas, les décision à prendre seraient l'intervalle de nos 4 variables. Par exemple, si la hauteur d'un objet est supérieure à 1m et sa largeur est comprise entre 2 et 3m alors il y a de fortes chances que l'objet en question soit un meuble. Dans notre cas, l'arbre est un plus complexe pour déterminer les catégories.

FIGURE 6 – Schéma de l'arbre de décision pour le problème

Cet arbre va permettre de trier les données en fonction des quatre variables renseigner pour se rapprocher au maximum d'une catégorie. Pour le calcul de score (précision) du réseau, nous considérons que pour un exemple, si le réseau n'a pas trouvé la bonne étiquette alors sa précision sur cet exemple sera de 0. Nous ferons ensuite la moyenne des précisions pour tous les exemples du dataset de test. Si nous séparons le dataset en 77/33 (77% du dataset est utilisé pour

l'entraînement, 33% pour le test) et en réalisant la prédiction sur un arbre, nous obtenons une précision de **74.07%** sur le dataset de test. Donc sur nos 270 données de test (tirées aléatoirement dans le dataset mélangé), nous en avons prédites correctement 200. La question est donc de savoir, sur quelles valeurs nous nous sommes trompés. Si nous affichons la matrice de confusion, nous remarquons que l'erreur se situe majoritairement entre la prédiction d'un objet 'mlp' et 'deco'. Les objets de type 'meuble' sont quant à eux mieux prédits de part leurs dimensions plus significatives que leurs pairs. Cependant pour un modèle aussi simple qu'un arbre de décision, une précision de 76% signifie que le classifieur est assez performant sur nos données.

True \ Pred	mlp	deco	meuble
mlp	70	22	0
deco	29	44	11
meuble	1	7	86

TABLE 1 – Matrice de confusion sur le dataset de test = 270 échantillons, un Arbre de décision.

2.3 Random Forest

Comme l'arbre de décision est performant, nous décidons d'utiliser les random forests (forêts aléatoires). Ces forêts sont un ensemble d'arbres de décision avec un paramètre aléatoire qui va permettre de sectionner qu'un certain nombre de variables pour construire son arbre. Lors de la prédiction, chaque arbre de la forêt devra voter pour la prédiction qu'il attribut à l'objet à prédire. Le choix de la réponse sera la prédiction qui aura le plus de votes. Plus nous augmentons notre nombre d'arbres, plus nous avons donc de 'votants' et plus nous pouvons fournir une réponse stable. En utilisant des Random Forests, nous nous rendons compte qu'entre 200 et 250 arbres (estimators) nous n'améliorons plus réseau. Nous décidons donc de prendre 250 arbres de décision et de changer le paramètre de séparation des données pour augmenter l'efficacité de notre réseau.

Pourcentage Train/Test	Outliers (52 valeurs)	Précision sur Test (%)
60/40	Avec	0.794
60/40	Sans	0.784
70/30	Avec	0.804
70/30	Sans	0.794
75/25	Avec	0.8178
75/25	Sans	0.782
78/22	Avec	0.8181
78/22	Sans	0.801
80/20	Avec	0.794
80/20	Sans	0.792

TABLE 2 – Performance sur RF(250) en fonction du retrait des outliers et de la séparation train/test du dataset

Comme le dataset a peu de données (900), nous remarquons que retirer les outliers (ce qui corresponds à 52 entrées ici) baisserait le nombre de données d'apprentissage et donc la précision de notre modèle. Notre modèle ici est simple mais souffre d'un manque de données importantes pour distinguer 'mlp' et 'deco'. En nous basant sur les performances des Random Forests en fonction de la séparation du dataset, nous utiliserons alors une séparation de 78% du dataset pour l'entraînement et 22 pour le test. Notre modèle final de prédiction a donc une précision de 81,82% sur le dataset de test.

2.4 Exemple en deep learning

Pour augmenter la complexité de notre modèle, nous décidons d'essayer un modèle plus complexe en utilisant un petit réseau de neurones profond. Nous utilisons Keras et construisons un simple réseau séquentiel avec 4 couches Denses de 8 unités. Cela signifie qu'a chaque couche, il y aura 8 neurones où chaque neurone sera connecté avec tous les neurones précédents. Lorsque nous exécutons le code, nous remarquons que la précision sur le dataset de test ne dépasse même pas la précision d'un arbre de décision. De plus, nous remarquons une très grande instabilité de la précision dû au peu de données que nous avons. L'augmentation du nombre d'epochs ou la baisse de la taille des batchs n'améliore pas notre réseau. Nous pensons donc qu'utiliser un réseau de deep est trop complexe pour la quantité de données que nous avons. Nous restons donc sur le classifieur des Random Forests pour la suite de nos prédictions.

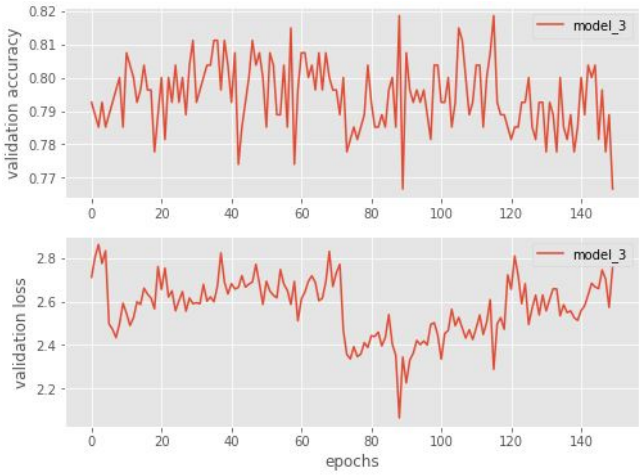


FIGURE 7 – Exemple d'un réseau simple de deep

3 API Flask

Si nous souhaitons faire une prediction pour une hauteur de 10, une largeur de 10, une profondeur de 10 et un poids de 10, alors nous nous rendons à l'adresse : "127.0.0.1 :5000/predict ?height=10weight=10width=10depth=10" en suivant la route 'predict' de notre api. La réponse de l'api sera sous la forme d'un dictionnaire et donnera la catégorie de l'objet prédite.