TD : Calcul des prix des véhicules - MLP

1. Pré-traitement des datasets

- Télécharger les données correspondant aux prix de véhicules.
- Affichez la taille et les types du data, supprimer la colonne *Unnamed*.
- Y a-t-il des doublons, supprimez-les. Supprimez également la colonne
- Combien y-a-t-il de données différentes correspondant à la variable *Cars Names*. Pensez-vous qu'il faille conserver cette colonne.
- Affichez via la fonction *describe()* que constatez-vous, pensez-vous qu'il faille traiter les données en l'état.
- Affichez les différentes valeurs de la colonne *Company Names*, que constatez-vous.
- Grâce à la fonction *apply()* des transformer les données en minuscules *lower()* et supprimez les blancs superflus *strip()*.
- Effectuer le même travail avec les colonnes *Engines* et *Fuel Types*.
- Combien y-a-t-il de valeurs différentes pour la colonne *Fuel Types*.
- Utilisez la commande *replace()* pour transforme les *hybrid / plug-in* et *plug-in hybrid* en *hybrid*.
- Affichez comment se répartissent les prix des véhicules. Que constatez-vous, peut-on conserver ces valeurs telles quelles.
- Transformer les prix en utilisant la fonction *np.log1p* de *numpy*, qui transforme les données x en log(1+x).
- Quelles sont les valeurs avec une seule occurrence pour les colonnes *objet*. Vous ne conserverez qu'une seule valeur par colonne.
- Appliquer la fonction get_dummies() permettant de transformer les données objets en colonnes différentes, et supprimez les colonnes que vous avez identifiées à la question précédente.

2. Préparation des données pour l'apprentissage

- Utiliser maintenant un model *StandardScaler()* pour normaliser les colonnes exceptée les *boolean*. Pourquoi réaliser cette opération.

- Découper maintenant le *dataframe* en un jeu de *train* et un jeu de *test* (ne séparez pas dans un premier temps les X et y).
- Certaines opérations traitent mieux les données lorsque les index se suivent. Utilisez la commande *reset_index()* pour réindexer le *train* et le *test*.
- Quelle est la taille du jeu de train. Lorsque le jeu de données contient des exemples rares on peut augmenter le jeu de données (*oversampling*). Utiliser la commande *concat()* pour augmenter le train. Vous dupliquerez quatre fois le jeu *train* d'origine.
- A partir du jeu du nouveau train et du test, créer les data *X_train, X_test, y_train, y_test*.

3. Modèle MLP - regression

- Créer un premier modèle séquentiel, qui doit recevoir en tant que première couche une couche Input avec un *shape(N,)* ou *N* est le nombre de *features* du jeu de train.
- Proposé un modèle composé de deux couches cachées *Dense* de 32 et 64 neurones et d'une couche de sortie. Combien doit-il y avoir de neurones en sortie.
- La sortie doit expliquer le prix, il n'est donc pas nécessaire de disposer d'une fonction de régularisation (*None* ou *linear*).
- Affichez via *summary()*, comment sont calculé les paramètres.
- Définissez maintenant le paramètres du modèle (*optimizer*, *loss* et *metrics*). Pourquoi doit-on utiliser pour la *loss mse*, et pour la *metrics RootMeanSquaredError (ou RMSE).*
- Entraînez le modèle sur 200 *epochs* avec des *batchs* de 256 et un *validation_split* de 20%.
- Affichez les courbes *loss*, *val_loss*, *root_mean_squared_error*, et *val_root_mean_squared_error*. Commentez les résultats.
- Calcule les prédictions et affichez via un *scatterplot* le lien entre les valeurs de test et les prédictions. Commentez les résultats.
- Le modèle peut être amélioré en ajoutant une couche de normalisation *BatchNormalization* a la sortie du modèles *Dense* dans les couches

cachées. Dans ce cas les fonctions d'activation doivent être placées après les normalisations.

- Affichez le modèle, combien y-a-t-il de paramètre par neurones de la couche dense dans les *BatchNormalization.*
- Évaluer le modèle et commentez les résultats obtenus précédement.