TP1 – Régression linéaire par apprentissage

Introduction à l'apprentissage profond
M2 informatique, Université Paris Cité, 2024-2025

Vous trouvez joint à ce TP un fichier `tp1-data.csv` contenant, à chaque ligne, deux valeurs entières `x` et `y`, séparées par une virgule.
L'objectif de cet exercice est de développer un modèle linéaire qui prédit la seconde valeur en fonction de la première. Ce sera également l'occasion de vous familiariser avec les opérations de base sur les `array`s de `numpy`.
La première ligne de votre fichier sera `import numpy as np`. Vous pourrez ensuite appeler des fonctions de la bibliothèque `numpy` comme `np.array([1,2,3], dtype=float64)` ou `np.mean(xs)`.

La droite définie par les paramètres ϕ_0 et ϕ_1 est donnée par :

$$f(x, \phi_0, \phi_1) = \phi_0 + \phi_1 \cdot x.$$

Le modèle cible est celui qui minimise la fonction de perte des moindres carrés. Pour n échantillons $(x_1, y_1), \ldots, (x_n, y_n)$, la *fonction de perte des moindres carrés* est définie par :

$$L(\phi_0, \phi_1) = \sum_{i=1}^{n} (f(x_i, \phi_0, \phi_1) - y_i)^2.$$

Note : Pour ce cas très simple, implémenter un algorithme d'apprentissage est en réalité excessif, et un raccourci mathématique est possible (voir la section "Solution analytique optimal" à la fin de ce TP). L'interêt de cet exercice est de développer un modèle d'apprentissage dans un cadre simple, donc n'utilisez pas ce raccourci au début.

Importation et visualisation des données

1. Écrivez une fonction qui lit les données dans le fichier `tp1-data.csv` et retourne deux tableaux `numpy` de même longueur (type `np.float64`), contenant respectivement toutes les valeurs `x` et toutes les valeurs `y`. Vous pouvez par exemple utiliser la fonction `loadtxt` de `numpy`.
2. Visualisez les données avec un diagramme de dispersion. Par exemple, `import plotly.express as px` et utilisez `px.scatter(x=xs, y=ys).show()`. Vous pouvez également utiliser `matplotlib` ou toute autre bibliothèque de visualisation.

Note : Ce cours suppose que vous êtes capable d'effectuer des tâches simples comme le traitement et la visualisation de données et modèles. Sinon, prenez du temps pendant cette première semaine de cours pour suivre des tutoriels sur des bibliothèques comme `pandas` ou `plotly`.

Prédiction et perte

3. Implémentez une fonction d'inférence `predict(xs, phis)` qui prend un tableau de valeurs `x` et un tableau contenant ϕ_0 et ϕ_1, et retourne un tableau de prédictions pour les valeurs `y`.
4. Implémentez la fonction de perte `loss(y_pred, y_true)` qui prend un tableau de valeurs prédites `y_pred` et un tableau de valeurs réelles `y_true`, et calcule la somme des pertes quadratiques.

Apprentissage manuel

5. Initialisez les paramètres ϕ_0 et ϕ_1 à `600` et `10`, respectivement. Créez un graphique contenant le nuage de points des données et la droite prédite, puis calculez la perte.
6. En observant le graphique, pouvez-vous deviner une meilleure valeur pour les paramètres ? Quelle est la perte ?

Calcul des gradients

Pour trouver les paramètres ϕ_0 et ϕ_1 qui ajustent bien les données, on peut répéter la question 6 de nombreuses fois à la main. L'idée de l'apprentissage automatique est d'automatiser ce processus. Pour cela, nous avons besoin de savoir comment la fonction de perte varie lorsque les paramètres changent, ce qui est mesuré par le *gradient*. En général, le gradient est une matrice contenant de nombreuses dérivées partielles ; nous verrons sa définition plus tard dans le cours. Pour ce modèle simple, les deux dérivées pertinentes peuvent être calculées ainsi:

$$\frac{\partial L}{\partial \phi_0} = 2\sum_{i=1}^{n}(f(x_i, \phi_0, \phi_1) - y_i).$$

$$\frac{\partial L}{\partial \phi_1} = 2\sum_{i=1}^{n}(f(x_i, \phi_0, \phi_1) - y_i) \cdot x_i.$$

7. Écrivez une fonction `loss_gradient` qui prend en entrée les tableaux `xs` et `ys`, ainsi que le tableau des paramètres `phis`, et retourne $\frac{\partial L}{\partial \phi_0}$ et $\frac{\partial L}{\partial \phi_1}$.

Note : Pour des modèles plus complexes, il est difficile de calculer le gradient analytiquement. Plus tard dans ce cours, nous verrons des méthodes automatiques pour calculer ou approcher le gradient.

8. Écrivez une fonction `step` qui effectue une étape de *descente de gradient* : étant donné les tableaux `xs`, `ys` et `phis`, elle retourne les nouvelles valeurs ϕ'_0 et ϕ'_1 définies par :

$$\phi'_k \leftarrow \phi_k - \alpha \cdot \frac{\partial L}{\partial \phi_k},$$

où α est une constante appelée *taux d'apprentissage*. Utilisez $\alpha = 10^{-6}$ (`alpha = 1e-6` en Python).

9. Écrivez une *boucle d'apprentissage* qui effectue `10` étapes de descente de gradient. Visualisez le modèle (la droite) à chaque étape, et après l'entraînement, visualisez la fonction de perte.

10. En quoi le choix des paramètres initiaux influence-t-il les performances de votre apprentissage ? Expérimentez avec de meilleurs ou pires paramètres initiaux.

Solution analytique optimal

11. Consultez la documentation `numpy` pour la fonction `numpy.linalg.lstsq`. Utilisez-la pour trouver la **solution analytique** du problème de régression linéaire. Comparez les valeurs des paramètres et la perte obtenues avec celles trouvées par l'apprentissage.

Vérification des gradients

Pour une fonction h de variables d'entrée x_1 et x_2, la dérivée partielle de h en (a, b) par rapport à la variable x_1 peut être approximée par :

$$\frac{\partial h(x_1, x_2)}{\partial x_1}(a, b) \approx \frac{h(a + \epsilon, b) - h(a - \epsilon, b)}{2\epsilon}$$

où ϵ est un petit nombre.

12. Vérifiez que votre calcul du gradient à la question 7 est correct, en implémentant une fonction `check_gradient` qui compare les dérivées calculées avec celles approximées par l'équation ci-dessus. Utilisez $\epsilon = 10^{-7}$.

Autres données

13. Utilisez maintenant le fichier `tp1-data2.csv`, contenant cette fois des valeurs flottantes. Dans quelle mesure pouvez-vous réutiliser le code écrit précédemment pour effectuer la régression linéaire sur ce nouveau jeu de données ? Refactorez votre code pour le rendre plus modulaire si nécessaire.