

Objectifs du TP :

- implémenter et analyser l'algorithme de régression linéaire par moindres carrés,
- utiliser scikit-learn pour tester et comparer deux algorithmes de régression régularisée, Ridge et Lasso, extension de la régression par moindres carrés,
- tester les algorithmes sur des données simulées et des données réelles.

1 Régression linéaire par moindres carrés

L'algorithme de régression par moindres carrés, dans sa version standard, est un algorithme de régression *linéaire*. Les données d'apprentissage utilisées dans cette section s'écrivent sous la forme $S = \{(x_i, y_i)\}_{i=1}^n$ avec $x_i \in \mathbb{R}^d$ (d est le nombre d'attributs des données d'entrée) et $y_i \in \mathbb{R}$.

A partir des données $\{(x_i, y_i)\}_{i=1}^n$, l'objectif est d'apprendre un vecteur $w \in \mathbb{R}^d$ tel que $y_i \approx \langle w, x_i \rangle$ pour tout $x_i \in S$. $\langle w, x \rangle := \sum_{j=1}^d w^{(j)} x^{(j)}$ est le produit scalaire entre les deux vecteurs w et x , avec $w^{(j)}$ et $x^{(j)}$ les j -èmes composantes des vecteurs w et x .

L'idée de l'algorithme de régression linéaire par moindres carrés est de résoudre le problème d'optimisation suivant :

$$\arg \min_{w \in \mathbb{R}^d} \sum_{i=1}^n (y_i - \langle w, x_i \rangle)^2.$$

Une forme analytique de la solution de ce problème est donnée par la formule suivante :

$$w = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top y,$$

avec $\mathbf{X} \in \mathbb{R}^{n \times d}$ la matrice contenant les données $(x_i)_{i=1}^n$ et $y \in \mathbb{R}^n$ le vecteur de labels $(y_i)_{i=1}^n$.

Algorithme Régression linéaire par moindres carrés

Entrée : une liste S de données d'apprentissage, $(x_1, y_1), \dots, (x_n, y_n)$ où $x_i \in \mathbb{R}^d$ et $y_i \in \mathbb{R}$,

Sortie : le vecteur de pondération w .

1. Ajouter le vecteur $\mathbf{1}$ à la matrice $\mathbf{X} \in \mathbb{R}^{n \times d}$ contenant les données $(x_i)_{i=1}^n$,
 2. Calculer $w = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top y$, avec $y = (y_i)_{i=1}^n \in \mathbb{R}^n$,
 3. Retourner w .
-

1) Implémentez l'algorithme de régression linéaire par moindres carrés décrit ci-dessus.

2) Le fichier `dataRegLin2D` contient un jeu de données bi-dimensionnelles. Chaque exemple de ce jeu de données est constitué de deux valeurs réelles (2D) associées à une étiquette de valeur réelle aussi. Utilisez la commande `loadtext` de `numpy` pour accéder aux données sous python. Affichez les données sur un graphique 3D (trois axes : $x^{(1)}$, $x^{(2)}$ et y). La dépendance entre les étiquettes y_i et les données x_i est-elle linéaire ? Représentez aussi sur deux graphiques 2D les points y_i et $x_i^{(1)}$ et y_i et $x_i^{(2)}$. Que remarquez-vous ?

3) Appliquez l'algorithme de régression linéaire par moindres carrés avec les différentes configurations d'entrée/sortie décrites dans 2) et représentez sur les mêmes graphiques les données et les solutions de régression obtenues.

4) Ecrire une fonction qui permet de prédire le label y_{test} d'une donnée x_{test} utilisant un vecteur de pondération w appris avec l'algorithme de régression par moindres carrés et calculer l'erreur de prédiction sur les données d'apprentissage.

2 Régression linéaire avec Scikit-learn

L'objectif de cette partie est d'implémenter et analyser des algorithmes de régression utilisant Scikit-learn.

2.1 sur les mêmes données simulées

- 1) Utilisez la fonction `LinearRegression` de `sklearn.linear_model` pour appliquer l'algorithme de régression par moindres carrés sur les données simulées décrites dans 1.2.
- 2) Calculez l'erreur de prédiction sur les données d'apprentissage et comparez les résultats avec ceux obtenus dans 1.4. Pour cela, utilisez la fonction `mean_squared_error` de `sklearn.metrics`.

2.2 sur des données réelles

- 3) Appliquez une analyse par régression linéaire par moindres carrés sur les jeux de données réelles `boston` et `diabetes`, disponibles dans `sklearn.datasets`.

2.3 et avec régularisation : Ridge et Lasso

La régression ridge et lasso sont des extensions de la régression linéaire par moindres carrés permettant d'éviter le risque de sur-apprentissage. L'idée est d'ajouter une pénalisation au problème de régression par moindres carrés :

$$\arg \min_{w \in \mathbb{R}^d} \sum_{i=1}^n (y_i - \langle w, x_i \rangle)^2 + \lambda \Omega(w),$$

où $\lambda \in \mathbb{R}$ est un paramètre de régularisation et $\Omega(w) = \|w\|_2^2$ pour la régression ridge et $\Omega(w) = \|w\|_1$ pour le Lasso.

- 4) Appliquez la régression Ridge et la régression Lasso sur le jeu de données `boston` avec $\alpha = 1.0$. Les deux fonctions `Ridge` et `Lasso` se trouvent dans `sklearn.linear_model`. Affichez et comparez les deux solutions obtenues.
- 5) Calculez les erreurs de prédiction sur les données d'apprentissage obtenues avec les régressions Ridge et Lasso. Comparez les résultats avec ceux obtenus par la régression par moindre carrés.
- 6) Le choix du paramètre α est primordiale pour avoir des résultats de prédiction optimaux. Une façon de procéder pour trouver une bonne valeur α est d'utiliser la méthode de cross-validation sur une grille de valeurs (voir la fonction `GridSearchCV`). Essayez les lignes de codes ci-dessous pour déterminer les valeurs de α permettant d'avoir les meilleurs taux de prédiction.

```
from sklearn.grid_search import GridSearchCV
alphas = np.logspace(-3, -1, 20)
for Model in [Ridge, Lasso]:
    gscv = GridSearchCV(Model(), dict(alpha=alphas), cv=5).fit(X, y)
    print(Model.__name__, gscv.best_params_)
```