Exercice 1 — Revue de code — 9 points

1. On a demandé à un data scientist d'estimer l'erreur de généralisation de la ridge régression sur des données de *Boston houssing* normalisées avec un paramètre $\lambda = 1$,

$$\widehat{\beta_r} = \underset{\beta \in \mathbf{R}^p}{\arg\min} \ \frac{1}{2}\|y - X\beta\|^2 + \frac{\lambda}{2}\|\beta\|^2$$

   Le code suivant à été produit. Qu'en pensez vous ?

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets

boston_ds = datasets.load_boston()
X = boston_ds.data
y = boston_ds.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42)
m_X = X_train.mean()
s_X = X_train.std()
X_n = (X_train - X_train.mean(axis=0))/X_train.std(axis=0)
X_tn = (X_test - X_train.mean(axis=0))/X_train.std(axis=0)
lam=1.0
n,p = np.shape(X_n)
beta = np.linalg.inv(X_n.T@X_n + lam*np.eye(p)) @ (X_n.T@y_train)
nt,p = np.shape(X_tn)
err = np.sum((y_test - (X_tn@beta + y_train.mean()))**2)/nt
```

2. Si vous aviez UNE recommendation à faire à l'auteur de ces lignes de code concernant sa méthodologie en apprentissage automatique (machine learning), quelle serait-elle ?

```
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

random_state = 42
X, y = make_regression(random_state=random_state, n_features=1, noise=1)
X_train, X_test, y_train, y_test = train_test_split(
                                X, y, test_size=0.4, random_state=random_state)
scaler = StandardScaler()
X_train_transformed = scaler.fit_transform(X_train)
model = LinearRegression().fit(X_train_transformed, y_train)
mean_squared_error(y_test, model.predict(X_test))
```

3. Si vous aviez UNE recommandation à faire à cet autre auteur de ces lignes de code concernant sa méthodologie en apprentissage automatique (machine learning), quelle serait-elle ?

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

n_samples, n_features, n_classes = 200, 10000, 2
rng = np.random.RandomState(42)
X = rng.standard_normal((n_samples, n_features))
y = rng.choice(n_classes, n_samples)
X_selected = SelectKBest(k=25).fit_transform(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, random_state=42)
gbc = GradientBoostingClassifier(random_state=1)
gbc.fit(X_train, y_train)
GradientBoostingClassifier(random_state=1)
y_pred = gbc.predict(X_test)
accuracy_score(y_test, y_pred)
```

Exercice 2 — Êtes-vous meilleur que chatGPT ? — 5 points

Comme chatGPT, répondez aux questions suivantes en 10 à 20 lignes.

1. Comment gérer le sur-apprentissage et le sous-apprentissage en apprentissage automatique ?
2. Quand on utilise le Lasso, faut-il normaliser les données ? Justifiez votre réponse.
3. Quand doit-on utiliser une méthode d'autoML ?
4. Qu'est-ce que la notion de pipeline pour Sklearn, quel est son intérêt ?
5. Quelle est la meilleure méthode pour construire un système de recommandation (et pourquoi) ?

Exercice 3 — Calculs non reinaux mais matriciels — 6 points

Soit W une matrice diagonale de taille n et de terme général w_i.

$$W = \begin{pmatrix} w_1 & 0 & \dots & 0 & \dots & \dots & 0 \\ 0 & w_2 & \dots & 0 & \dots & \dots & 0 \\ \vdots & \dots & \ddots & \ddots & \ddots & \dots & \vdots \\ 0 & \dots & 0 & w_i & 0 & \dots & 0 \\ \vdots & \dots & \ddots & \ddots & \ddots & \dots & \vdots \\ 0 & \dots & \dots & 0 & \dots & w_{n-1} & 0 \\ 0 & \dots & \dots & 0 & \dots & 0 & w_n \end{pmatrix}$$

1. Donner la forme du terme général f_i du vecteur $\mathbf{f} = W\mathbf{e}$ en fonction des w_i et des e_i, où $\mathbf{e}$ est un vecteur de $\mathbb{R}^n$ de terme général e_i.
2. Donner également la forme générale de $\mathbf{e}^\top W\mathbf{e}$ en fonction des w_i et des e_i.
3. Pour des vecteurs $\mathbf{x} = (x_1, \dots, x_n)^\top$, $\mathbf{y} = (y_1, \dots, y_n)^\top$ et $\boldsymbol{w} = (w_1, \dots, w_n)^\top$ donnés, calculer la solution du problème suivant :

$$\min_{a,b} \; J(a,b) \qquad \text{avec } J(a,b) = \tfrac{1}{2} \sum_{i=1}^{n} w_i \big(y_i - (a + bx_i)\big)^2 .$$

4. Calcul matriciel :
   a) Écrire matriciellement $J(\alpha)$ avec $\alpha = (a,b)^\top$ et en fonction des vecteurs $\mathbf{x}, \mathbf{y}, \mathbb{1}$ et de la matrice W (où $\mathbb{1} = (1, \dots, 1)^\top$ est un vecteur de 1 de taille n).
   b) En déduire $\nabla_\alpha J$, le gradient de J par rapport à α.
   c) Donner l'expression de α optimal (solution du problème de minimisation) en fonction de $\mathbf{x}$, $\mathbf{y}, \mathbb{1}$ et W.
   d) Donner le code python permettant de résoudre ce problème, les vecteurs $\mathbf{x} = (x_1, \dots, x_n)^\top$, $\mathbf{y} = (y_1, \dots, y_n)^\top$ et $\boldsymbol{w} = (w_1, \dots, w_n)^\top$ étant donnés.

Exercice 4 — Revue de code — 9 points

1. On a demandé à un data scientist d'estimer l'erreur de généralisation de la ridge régression sur des données de *Boston houssing* normalisées avec un paramètre $\lambda = 1$,

$$\widehat{\beta_r} = \arg\min_{\beta} \ \frac{1}{2}\|y - X\beta\|^2 + \frac{\lambda}{2}\|\beta\|^2$$

   Le code semble implémenter une régression Ridge en utilisant une matrice de design normalisée. Toutefois, il y a quelques points qui pourraient être améliorés :

   1. Le code ne normalise pas correctement les données de test. Les données de test doivent être normalisées en utilisant les moyennes et les écarts-types calculés sur l'ensemble d'entraînement, pas sur l'ensemble de test. Le code actuel utilise les moyennes et les écarts-types calculés sur l'ensemble d'entraînement pour normaliser les données de test, ce qui peut conduire à des résultats imprécis.
   2. Le code calcule l'erreur de généralisation comme la somme des carrés des erreurs sur l'ensemble de test, mais il ne la divise pas par le nombre d'observations. Il est recommandé de diviser la somme des carrés des erreurs par le nombre d'observations pour obtenir une valeur moyenne des erreurs.
   3. Le code ne vérifie pas si la matrice de design est inversible. Si la matrice de design est singulière (c'est-à-dire si certaines colonnes sont linéairement dépendantes), la régression Ridge ne pourra pas être effectuée. Il est recommandé de vérifier si la matrice de design est inversible avant de l'utiliser dans l'algorithme.
   4. Le code ne vérifie pas si les données sont correctement mises en forme.

2. Le code suivant à été produit. Qu'en pensez vous ?
   Une recommandation que je pourrais faire concernant cette méthodologie en apprentissage automatique serait de s'assurer de disposer d'un ensemble de données de validation pour évaluer les performances du modèle. Actuellement, le code utilise un ensemble de données de test uniquement pour évaluer les performances du modèle, ce qui peut conduire à une surestimation des performances du modèle sur de nouvelles données.
   Pour éviter cela, il peut être utile de séparer votre ensemble de données en trois parties : un ensemble d'entraînement, un ensemble de validation et un ensemble de test. Vous pouvez utiliser l'ensemble d'entraînement pour entraîner votre modèle, l'ensemble de validation pour ajuster les hyperparamètres du modèle et l'ensemble de test pour évaluer les performances du modèle final. Cela peut vous aider à obtenir une évaluation plus précise des performances du modèle sur de nouvelles données.
3. Si vous aviez UNE recommandation à faire à cet autre auteur de ces lignes de code concernant sa méthodologie en apprentissage automatique (machine learning), quelle serait-elle ?

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

n_samples, n_features, n_classes = 200, 10000, 2
rng = np.random.RandomState(42)
X = rng.standard_normal((n_samples, n_features))
y = rng.choice(n_classes, n_samples)
X_selected = SelectKBest(k=25).fit_transform(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, random_state
    =42)
gbc = GradientBoostingClassifier(random_state=1)
gbc.fit(X_train, y_train)
GradientBoostingClassifier(random_state=1)
y_pred = gbc.predict(X_test)
accuracy_score(y_test, y_pred)
```

Une recommandation que je pourrais faire concernant cette méthodologie en apprentissage automatique serait d'utiliser une méthode de validation croisée pour évaluer les performances du modèle. La validation croisée consiste à diviser votre ensemble de données en plusieurs sous-ensembles, entraîner votre modèle sur chacun d'entre eux et évaluer les performances du modèle en utilisant le sous-ensemble restant. Cela peut vous aider à obtenir une évaluation plus précise des performances du modèle, car cela permet de prendre en compte plus de données dans l'évaluation.
La bibliothèque Sklearn de Python fournit plusieurs fonctions qui facilitent l'utilisation de la validation croisée, comme cross_val_score et GridSearchCV. Vous pouvez utiliser ces fonctions pour automatiser le processus de validation croisée et obtenir une évaluation plus précise des performances de votre modèle.

Exercice 5 — Êtes-vous meilleur que chatGPT ? — 5 points

1. Comment gérer le sur-apprentissage et le sous-apprentissage ?
   Le sur-apprentissage et le sous-apprentissage sont deux problèmes courants en apprentissage automatique (machine learning).
   Le sur-apprentissage se produit lorsque votre modèle est entraîné de manière trop précise sur un ensemble de données donné et ne parvient pas à généraliser ses prédictions sur de nouvelles données. Cela peut se produire lorsque votre modèle a trop de paramètres ou lorsque vous entraînez votre modèle sur un petit ensemble de données. Pour éviter le sur-apprentissage, vous pouvez utiliser une technique de régularisation, comme le régulariseur L2 ou le dropout, pour limiter la complexité du modèle et l'aider à généraliser mieux ses prédictions. Vous pouvez également augmenter la taille de votre ensemble de données d'entraînement pour que votre modèle ait plus d'exemples à partir desquels apprendre.
   Le sous-apprentissage se produit lorsque votre modèle n'a pas assez de données ou de puissance de calcul pour apprendre la relation entre les entrées et les sorties. Dans ce cas, votre modèle ne sera pas en mesure de faire de bonnes prédictions sur de nouvelles données. Pour éviter le sous-apprentissage, vous pouvez augmenter la taille de votre ensemble de données d'entraînement, utiliser un modèle de plus grande complexité ou ajouter plus de puissance de calcul (par exemple, en utilisant un GPU au lieu d'un CPU).
2. Quand on utilise le Lasso, faut-il normaliser les données ?
   Il est souvent recommandé de normaliser les données lorsque vous utilisez un régulariseur tel que le Lasso, car cela peut améliorer les performances du modèle et rendre l'optimisation du modèle plus stable.
   La normalisation consiste à mettre toutes les variables sur le même échelle, généralement en les centrant et en les réduisant. Cela peut être utile car les différentes variables dans votre ensemble de données peuvent avoir des échelles très différentes, ce qui peut rendre difficile pour le modèle de trouver un bon compromis entre les différentes variables. En normalisant les données, vous pouvez vous assurer que toutes les variables ont un poids équivalent dans le modèle.
   Cependant, il y a certaines situations où la normalisation des données peut ne pas être nécessaire. Par exemple, si toutes les variables ont déjà une échelle similaire ou si vous utilisez un algorithme qui est insensible à l'échelle des variables (comme les arbres de décision), la normalisation des données peut ne pas être nécessaire. En fin de compte, cela dépend de votre modèle et de vos données. Il est recommandé de tester différentes approches et de voir ce qui fonctionne le mieux pour votre cas d'utilisation spécifique.
3. Quand doit-on utiliser une méthode d'autoML ?
   L'autoML (Automated Machine Learning) est une approche qui vise à automatiser certaines tâches fastidieuses et chronophages liées à l'apprentissage automatique, comme la sélection de modèles et le réglage de leurs hyperparamètres. L'autoML peut être utile dans les situations suivantes :
   Lorsque vous êtes débutant en apprentissage automatique et que vous voulez avoir une idée de ce qui fonctionne le mieux sans avoir à apprendre les détails de chaque modèle et de chaque algorithme.
   Lorsque vous avez peu de temps et que vous voulez obtenir rapidement des résultats sans avoir à passer trop de temps à essayer différents modèles et à ajuster les hyperparamètres.
   Lorsque vous avez un grand nombre de modèles à essayer et que vous ne voulez pas passer trop de temps à les optimiser manuellement.
   Lorsque vous voulez utiliser une approche "plug and play" et que vous êtes prêt à accepter un compromis sur les performances en échange d'un gain de temps.

Il est important de noter que l'autoML n'est pas toujours la meilleure approche et qu'il peut y avoir des cas où une approche manuelle est préférable. Par exemple, si vous avez une connaissance approfondie de votre domaine d'application et que vous savez quels modèles sont les mieux adaptés à vos données, il peut être plus efficace de les entraîner et de les régler manuellement.

4. Qu'est-ce que la notion de pipeline pour Sklearn, quel est son intérêt ?
   En apprentissage automatique, un pipeline est une série d'étapes de traitement des données qui sont exécutées les unes à la suite des autres. Cela peut inclure des étapes de prétraitement des données, de sélection de modèle et de réglage d'hyperparamètres.
   L'utilisation de pipelines est particulièrement utile lorsque vous avez plusieurs étapes de traitement des données et que vous voulez automatiser le processus de bout en bout. Cela peut vous aider à éviter les erreurs et à gagner du temps en évitant de devoir réécrire du code à chaque fois que vous voulez exécuter l'ensemble des étapes.
   La bibliothèque Sklearn de Python fournit un module Pipeline qui vous permet de créer et de gérer facilement des pipelines de traitement des données. Vous pouvez utiliser ce module pour créer un pipeline en spécifiant les différentes étapes que vous voulez exécuter et en les chainant les unes aux autres. Vous pouvez également utiliser le module GridSearchCV de Sklearn pour automatiser le réglage des hyperparamètres de votre modèle dans le cadre de votre pipeline.
   En résumé, l'utilisation de pipelines peut vous aider à automatiser et à simplifier le processus d'apprentissage automatique en regroupant toutes les étapes de traitement des données dans un seul environnement.
5. Quelle est la meilleure méthode pour construire un système de recommandation (et pourquoi) ?
   Il existe plusieurs approches pour construire un système de recommandation, et la meilleure méthode dépend de votre domaine d'application et de vos besoins spécifiques. Voici quelques méthodes couramment utilisées :
   Filtrage collaboratif : cette approche utilise les données sur les interactions des utilisateurs (par exemple, les articles qu'ils ont achetés ou aimés) pour construire un modèle de préférences des utilisateurs. Le modèle peut ensuite être utilisé pour recommander des éléments similaires aux éléments précédemment appréciés par l'utilisateur.
   Filtrage basé sur le contenu : cette approche utilise les caractéristiques des éléments que l'on veut recommander (par exemple, les métadonnées des films ou les mots clés des articles) pour trouver des éléments similaires.
   Méthodes hybrides : cette approche combine plusieurs méthodes de recommandation, comme le filtrage collaboratif et le filtrage basé sur le contenu, pour obtenir des résultats améliorés.
   Il n'y a pas de méthode "meilleure" en soi, mais certaines approches peuvent être plus adaptées à certaines situations que d'autres. Par exemple, le filtrage collaboratif peut être particulièrement utile lorsque vous avez beaucoup de données sur les interactions des utilisateurs, tandis que le filtrage basé sur le contenu peut être préférable lorsque vous avez peu de données sur les utilisateurs mais beaucoup de métadonnées sur les éléments à recommander. Il est recommandé de tester différentes approches et de voir ce qui fonctionne le mieux pour votre cas d'utilisation spécifique.

Exercice 6 — Calculs non reinaux mais matriciels — 6 points

1. Le terme général f_i du vecteur $\mathbf{f} = W\mathbf{e}$ est donné par $f_i = w_i \cdot e_i$.
2. La forme générale de $\mathbf{e}^\top W\mathbf{e}$ est donnée par $\sum_{i=1}^{n} w_i \cdot e_i^2$.
3. Pour résoudre le problème de minimisation, nous devons trouver les valeurs de a et b qui minimisent la fonction $J(a, b)$. Pour cela, nous pouvons utiliser la dérivée partielle de J par rapport à a et b et mettre ces dérivées à zéro. Nous obtenons alors les équations suivantes :

$$\sum_{i=1}^{n} w_i(y_i - (a + bx_i)) = 0$$

$$\sum_{i=1}^{n} w_i x_i(y_i - (a + bx_i)) = 0$$

   En résolvant ce système d'équations, nous obtenons la solution suivante :

$$a = \frac{\sum_{i=1}^{n} w_i y_i - b\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

$$b = \frac{\sum_{i=1}^{n} w_i x_i y_i - a \sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i x_i}$$