Université de Montpellier Année 2025-2026
HAX912X - Modèles linéaires généralisés

Examen - Vendredi 24 octobre 2025 - Durée 2h00
Tous documents autorisés, Outils numériques interdits

Exercice 1 (10 pts)

Le jeu de données House Prices : Advanced Regression Techniques (Kaggle) contient des informations détaillées sur les caractéristiques de maisons résidentielles situées à Ames, dans l'Iowa (États-Unis). Il comprend $1,460$ observations et 80 variables décrivant aussi bien des aspects numériques (surface habitable, superficie du terrain, année de construction, nombre de pièces, etc.) que catégoriels (type de quartier, matériau extérieur, qualité du garage, etc.). La variable à prédire est le prix de vente de la maison (SalePrice).
Le code qui suit met en œuvre des méthodologies sur ce jeu de données.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import RidgeCV

data = pd.read_csv("train.csv")
data.shape
(1460, 81)

missing_counts = data.isnull().sum()
print(missing_counts.sum())
7829

cols_to_drop = missing_counts[missing_counts > 200].index
data = data.drop(columns=list(cols_to_drop))
data.shape
(1460, 74)

data = data.dropna()
data.shape
(1338, 74)

X = data.drop(columns=["SalePrice", "Id"])
X.shape
(1338, 72)

y = data["SalePrice"]
y.shape
(1338,)
```

```
numerical_cols = X.select_dtypes(include=np.number).columns
categorical_cols = X.select_dtypes(exclude=np.number).columns
scaler = StandardScaler()
X_num = pd.DataFrame(
    scaler.fit_transform(X[numerical_cols]),
    columns=numerical_cols,
    index=X.index
)

encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
X_cat = pd.DataFrame(
    encoder.fit_transform(X[categorical_cols]),
    columns=encoder.get_feature_names_out(categorical_cols),
    index=X.index
)

X = pd.concat([X_num, X_cat], axis=1)
X.shape
(1338, 260)

X_train, X_test, y_train, y_test = train_test_split(X, y,
  test_size=0.2, random_state=42)

alphas = np.logspace(-3, 3, 20)
model1 = RidgeCV(alphas=alphas, cv=5)
model1.fit(X_train, y_train)
print(model1.alpha_)
497.7023564332114

y_pred = model1.predict(X_test)
print(np.sqrt(mean_squared_error(y_test, y_pred)))
27263.849831918353
print(r2_score(y_test, y_pred))
0.839711840063

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import copy

device = torch.device("mps")
X_train_t = torch.from_numpy(X_train.to_numpy().astype(np.float32))
y_train_t = torch.from_numpy(y_train.to_numpy().astype(np.float32)).unsqueeze(1)
X_test_t  = torch.from_numpy(X_test.to_numpy().astype(np.float32))
y_test_t  = torch.from_numpy(y_test.to_numpy().astype(np.float32)).unsqueeze(1)
X_test_t = X_test_t.to(device)
y_test_t = y_test_t.to(device)
train_ds = TensorDataset(X_train_t, y_train_t)
train_loader = DataLoader(train_ds, batch_size=64, shuffle=True, drop_last=False)
```

```
model2 = nn.Sequential(
            nn.Linear(X_train.shape[1], 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 32),
            nn.ReLU(),
            nn.Linear(32, 1)
        )
sum(p.numel() for p in model2.parameters())
103873

patience = 100
best_loss = float('inf')
wait = 0
best_state = None
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model2.parameters(), lr=1e-3)
model2.to(device)
for epoch in range(1,3000):
    model2.train()
    running = 0.0
    for xb, yb in train_loader:
        xb = xb.to(device, dtype=torch.float32)
        yb = yb.to(device, dtype=torch.float32)
        optimizer.zero_grad()
        pred = model2(xb)
        loss = criterion(pred, yb)
        loss.backward()
        optimizer.step()
        running += loss.item() * xb.size(0)
    train_loss = np.sqrt(running / X_train.shape[0])
    if epoch == 1 or epoch % 10 == 0 or epoch == 3000:
        model2.eval()
        with torch.no_grad():
            y_pred = model2(X_test_t)
            test_loss = np.sqrt(criterion(y_pred, y_test_t).item())
        print(f"Epoch {epoch:02d} | Train loss = {train_loss:.5f} |
          Test loss={test_loss:.5f}")
    if test_loss < best_loss:
        best_loss = test_loss
        best_state = copy.deepcopy(model2.state_dict())
        wait = 0
    else:
        wait += 1
        if wait >= patience:
            print(f"Stop at {epoch:02d} | Best test loss={best_loss:.5f}")
            break
```

```
Epoch 01 | Train loss = 205726.84799 | Test loss=190329.21037
Epoch 10 | Train loss = 123978.49342 | Test loss=89385.48034
Epoch 20 | Train loss = 39057.69737 | Test loss=30446.40353
Epoch 30 | Train loss = 36059.01995 | Test loss=28538.54881
Epoch 40 | Train loss = 34390.81269 | Test loss=27454.60020
Epoch 50 | Train loss = 33182.08970 | Test loss=27113.19679
Epoch 60 | Train loss = 32074.82779 | Test loss=26560.80360
Epoch 70 | Train loss = 31201.54457 | Test loss=26252.47265
Epoch 80 | Train loss = 30577.49765 | Test loss=25848.02228
Epoch 90 | Train loss = 29939.09000 | Test loss=25757.12779
Epoch 100 | Train loss = 29409.93884 | Test loss=25930.18565
Epoch 110 | Train loss = 28913.90038 | Test loss=25747.94003
Epoch 120 | Train loss = 28517.68247 | Test loss=25718.91631
Epoch 130 | Train loss = 28192.56219 | Test loss=25933.95797
Epoch 140 | Train loss = 27914.90338 | Test loss=25881.33907
Epoch 150 | Train loss = 27613.05954 | Test loss=25835.31722
Epoch 160 | Train loss = 27497.38071 | Test loss=25884.25436
Epoch 170 | Train loss = 27180.96120 | Test loss=26022.10629
Epoch 180 | Train loss = 26883.80446 | Test loss=26159.97187
Epoch 190 | Train loss = 26682.09135 | Test loss=26261.79583
Epoch 200 | Train loss = 26712.71132 | Test loss=26591.13837
Epoch 210 | Train loss = 26315.48065 | Test loss=26572.24657
Epoch 220 | Train loss = 26200.98589 | Test loss=26590.90972
Stop at 220 | Best test loss=25718.91631

if best_state is not None:
    model2.load_state_dict(best_state)

model2.eval()
with torch.no_grad():
    y_pred = model2(X_test_t)
    y_pred = y_pred.detach().cpu().numpy().ravel()

print(np.sqrt(mean_squared_error(y_test, y_pred)))
25718.915450848068
print(r2_score(y_test, y_pred))
0.8454203605651855
```

1 (2 pts) Analysez en détails la préparation des données réalisée avant l'apprentissage.

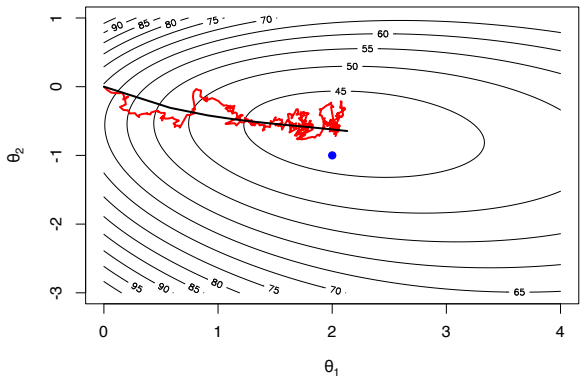
2 (3 pts) Décrire en détails comment le modèle 1 est construit et analyser les résultats.

3 (5 pts) Décrire en détails comment le modèle 2 est construit et analyser les résultats.

Exercice 2 (6 pts)

```
set.seed(2003)

N <- 100
X <- matrix(rnorm(N * 2), ncol = 2)
theta_true <- c(2, -1)

eta <- as.vector(X %*% theta_true)
p   <- 1/(1 + exp(-eta))
y   <- rbinom(N, size = 1, prob = p)

t1_seq <- seq(theta_true[1] - 2, theta_true[1] + 2, length.out = 120)
t2_seq <- seq(theta_true[2] - 2, theta_true[2] + 2, length.out = 120)

L <- matrix(0, nrow = length(t1_seq), ncol = length(t2_seq))
for (i in 1:length(t1_seq)) {
  for (j in 1:length(t2_seq)) {
    th <- c(t1_seq[i], t2_seq[j])
    eta <- as.vector(X %*% th)
    L[i, j] <- sum(log(1+exp(eta)) - y * eta)
  }
}

niter1 <- 5
theta.algo1 <- matrix(0, N*niter1 + 1, 2)
delta <- 0.08
for (k in 1:niter1) {
  indi <- sample(1:N, N)
  for (i in 1:N) {
    x_i   <- X[indi[i], ]
    y_i   <- y[indi[i]]
    eta_i <- sum(x_i * theta.algo1[(k-1)*N + i, ])
    p_i   <- 1/(1 + exp(-eta_i))
    grad  <- x_i * (p_i - y_i)
    theta.algo1[(k-1)*N + i + 1, ] <- theta.algo1[(k-1)*N + i, ] - delta * grad
  }
}

niter2 <- 50
theta.algo2 <- matrix(0, niter2 + 1, 2)
delta <- 0.02
for (k in 1:niter2) {
  eta_k <- as.vector(X %*% theta.algo2[k, ])
  p_k   <- 1/(1 + exp(-eta_k))
  # gradient plein lot: t(X) %*% (p - y)
  grad  <- t(X) %*% (p_k - y)
  theta.algo2[k + 1, ] <- theta.algo2[k, ] - delta * as.vector(grad)
}
```

```
contour(t1_seq, t2_seq, L,
        nlevels = 20,
        xlab = expression(theta[1]),
        ylab = expression(theta[2]))
lines(theta.algo1[, 1], theta.algo1[, 2], lwd = 2, col = "red")
lines(theta.algo2[, 1], theta.algo2[, 2], lwd = 2, col = "black")
points(theta_true[1], theta_true[2], pch = 19, col = "blue")
```



Décrire en détails les résultats obtenus : les données simulées et donc le modèle considéré (2 pts), ce que font les algorithmes 1 et 2 et ce que représente le graphique donné (4 pts).

Exercice 3 (4 pts)

1 (2 pts) Expliquer les résultats produits par la code R suivant.

```
library(klaR)
data(GermanCredit)
model1 <- glm(credit_risk~duration,family=binomial,data=GermanCredit)
model2 <- glm(credit_risk~1,data=GermanCredit,family=binomial)
model3 <- glm(credit_risk~.,data=GermanCredit,family=binomial)
```

2 (2 pts) Donner le code R permettant d'estimer par validation croisée à 2 ensembles répétée 10 fois les taux d'erreurs associés aux modèles considérés dans la question précédente.

Correction

Exercice 1

1 Le fichier `train.csv` contient 1460 individus et 81 variables. La variable cible est `SalePrice` et la variables `Id` est un identifiant. Il y a 7829 valeurs manquantes au total. Les variables avec plus de 200 valeurs manquantes sont supprimées, il reste alors 74 variables. Puis nous supprimons les individus pour lesquels il reste des valeurs manquantes qui réduit l'échantillon à 1338 individus.

La suppression complète des individus pour lesquels des valeurs sont manquantes peut introduire un biais d'échantillonnage et fait perdre de l'information ; une imputation, par exemple pour les quantitatives par la médiane et pour les variables qualitatives par la catégorie `Missing`) est souvent préférable.

Les variables quantitatives sont alors centrées réduites avec `StandardScaler`. Les variables qualitatives sont encodées par `OneHotEncoder(handle_unknown="ignore")`. Les variables qualitatives sont transformées en variables numériques. Chaque modalité d'une variable qualitative est convertie en colonne binaire (0/1). Pour une observation donnée, la colonne correspondant à sa modalité prend la valeur 1, les autres 0. Si une variable possède k modalités, l'encodeur crée k colonnes. Cela diffère de la pratique courante en régression linéaire ou logistique, où l'on supprime une modalité pour éviter la colinéarité parfaite entre colonnes. Ce problème est sans conséquence pour certains modèles, notamment le régression ridge et les réseaux de neurones.

Pour certains modèles, il est préférable de définir explicitement une modalité de référence, il faut alors utiliser la commande `OneHotEncoder(drop='first', handle_unknown='ignore')`. Cela crée $k-1$ colonnes par variable qualitative et évite la colinéarité parfaite entre variables explicatives. Le première modalité est alors la modalité de référence. Pour les réseaux de neurones ou les modèles régularisés, supprimer une modalité pour éviter la colinéarité peut éliminer des informations utiles et dégrader les performances prédictives.

Après concaténation, la matrice de design a 260 colonnes.

Le scaler et l'encodeur sont ajustés sur l'ensemble entier avant le split train/test, cela aurait du être fait sur le train uniquement, puis appliqués au test.

Il est alors effectué un découpage apprentissage/test 80/20.

`SalePrice` est en dollars et typiquement asymétrique à droite. Une transformation `log` aurait pu stabiliser la variance et aider certains modèles ; ici la cible est laissée brute, les métriques sont donc au sens des dollars (RMSE).

2 Le modèle 1 est un modèle de régression ridge avec validation croisée interne pour choisir la régularisation. Il est utilisé une grille d'alphas log-espacée de 10^{-3} à 10^{3} (20 valeurs) et une validation croisée à 5 ensembles. La procédure de validation croisée choisit $\alpha \approx 497,70$ Évaluation sur le test : RMSE ≈ 27263 et $R^2 \approx 0,840$. L'erreur quadratique moyenne correspond à une erreur de l'ordre de 27 mille dollars sur le niveau de prix.

3 Le modèle 2 est un réseau de neurones entièrement connecté recevant en entrée les 260 variables issues de la standardisation des variables numériques et de l'encodage des variables qualitatives. Il comporte trois couches cachées de 256, 128 et 32 neurones activées par des fonctions ReLU, suivies d'une couche de sortie linéaire à un neurone. L'optimisation est réalisée par l'algorithme Adam avec un taux d'apprentissage de $0,001$, une taille de lot de 64 et la fonction de perte MSE. L'entraînement est prévu pour un maximum de 3000 itérations et utilise un mécanisme d'arrêt précoce *early stopping* pour éviter le surapprentissage. Ce mécanisme repose sur la variable patience, fixée ici à 100 : après chaque évaluation, si la performance sur le jeu suivi, ici le test, ne s'améliore pas pendant 100 époques consécutives, l'entraînement est interrompu et le meilleur état du modèle est restauré. Le modèle atteint un RMSE d'environ 25700 et un R^2 de $0,845$ sur le test, soit une erreur moyenne d'environ 25 à 26 mille dollars sur le prix des maisons, légèrement meilleure que celle obtenue par la régression Ridge.

Le jeu de test est utilisé pour piloter l'arrêt précoce et sélectionner le meilleur modèle, ce qui introduit un biais et fausse l'évaluation finale ; d'autre part, la perte sur le test n'est calculée que toutes les dix époques (et non à chaque itération), ce qui peut faire manquer le véritable minimum de la courbe de performance. Il faut introduire un échantillon de validation distinct du test pour piloter l'arrêt précoce, et à réserver le test à une seule évaluation finale du modèle.

Exercice 2

Les données sont simulées suivant un modèle de régression logistique à deux covariables. Nous générons d'abord une matrice de covariables $\mathbf{X}$ de dimension $N = 100$ et à deux colonnes, chaque composante est simulée suivant une loi gaussienne centrée réduite. Nous fixons $\boldsymbol{\theta}_{\text{true}} = (2, -1)$ puis nous calculons pour chaque individu i la combinaison linéaire $\eta_i = x_{i,1}\theta_{\text{true},1} + x_{i,2}\theta_{\text{true},2}$ et enfin $p_i = \left[\dfrac{1}{1+e^{-\eta_i}}\right]$. La variable binaire y_i est simulée selon une loi Bernoulli $y_i \mid \mathbf{x}_i \sim \mathcal{B}(p_i)$. Ainsi, le modèle simulé est :

$$y_i \mid \mathbf{x}_i \sim \mathcal{B}(p_i), \quad \text{avec } \log\left(\frac{p_i}{1-p_i}\right) = \mathbf{x}_i^\top \boldsymbol{\theta}\,.$$

La matrice L calcule, pour un maillage de valeurs (θ_1, θ_2), l'opposé de la log-vraisemblance :

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left[\log(1+e^{\eta_i}) - y_i\eta_i\right]\,.$$

Nous souhaitons minimiser $L(\cdot)$ et donc maximiser la vraisemblance.

Algorithme 1 descente de gradient stochastique (SGD), pour chaque itération, nous parcourons les N observations dans un ordre aléatoire, nous calculons

$$\nabla_{\boldsymbol{\theta}} \left[\log(1+e^{\eta_i}) - y_i\eta_i\right] = \mathbf{x}_i(p_i - y_i)\,.$$

et nous mettons à jour le vecteur de paramètres selon la règle

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \delta\, \mathbf{x}_i(p_i - y_i)$$

avec un pas d'apprentissage $\delta = 0.08$.

Algorithme 2 descente de gradient (GD), à chaque itération, nous calculons le gradient sur tout l'échantillon

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \mathbf{X}^\top (\mathbf{p} - \mathbf{y})$$

puis nous mettons à jour

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \delta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}),$$

avec un pas $\delta = 0.02$.
Le graphique représente, en niveaux de gris les lignes de niveau de la fonction $L(\theta)$; le point bleu, la vrai valeur du paramètre $\boldsymbol{\theta}_{\text{true}} = (2, -1)$; la courbe rouge, les itérations successives de la descente de gradient stochastique (SGD) ; la courbe noire, la trajectoire de la descente de gradient (GD).

Exercice 3

1 Le jeu de données `GermanCredit` contient des informations sur des emprunteurs allemands : chaque observation décrit un individu avec plusieurs variables explicatives (durée du crédit, montant, emploi, âge, statut matrimonial, etc.) et une variable réponse binaire `credit_risk` indiquant si le risque de crédit est bon ou mauvais. Les trois modèles considérés sont des modèles de régression logistique

- model1 : modèle simple avec une seule variable explicative : la durée du crédit (`duration`),
- model2 : modèle sans variable explicative avec seulement l'intercept,
- model3 : modèle complet utilisant toutes les variables disponibles dans `GermanCredit`.

2

```
library(caret)
ctrl <- trainControl(method = "repeatedcv", number = 2,
     repeats = 10)
cv1 <- train(credit_risk ~ duration, data = GermanCredit,
             method = "glm", family = "binomial",
             trControl = ctrl)
GermanCredit$const <- 1
cv2 <- train(credit_risk ~ const-1, data = GermanCredit,
             method = "glm", family = "binomial",
             trControl = ctrl)
cv3 <- train(credit_risk ~ ., data = GermanCredit,
             method = "glm", family = "binomial",
             trControl = ctrl)
error_rates <- 1 - c(cv1$results$Accuracy, cv2$results$Accuracy,
     cv3$results$Accuracy)
names(error_rates) <- c("model1", "model2", "model3")
error_rates
```