

TD/TP 3 : compléments sur le modèle linéaire

UE Modèle linéaire

moi

Avertissement

Pensez à mettre votre nom dans l'entête du document.

Nous aurons besoin des packages ci-dessous pour faire les deux exercices. S'ils ne sont pas disponibles sur votre ordinateur, merci de les installer.

```
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.linear_model import RidgeCV, LassoCV, Lasso
from sklearn.preprocessing import StandardScaler
from scipy.linalg import eigh
import matplotlib.pyplot as plt
```

Exercice 1 : régression ridge

On s'intéresse à jeu de données portant sur l'endettement d'habitants des USA via leurs cartes de crédit. Les variables du jeu de données sont :

- **Income** : revenu annuel (en millier de dollars US)
- **Limit** : plafond de crédit (en dollars US)
- **Rating** : cote de solvabilité
- **Cards** : nombre de cartes de crédit
- **Age** : âge (en années)
- **Education** : nombre d'années d'instruction
- **Own** : propriétaire d'un logement
- **Student** : étudiant

- **Married** : statuts marital
- **Region** : région de vie
- **Balance** : dette moyenne sur la carte de crédit

L'objectif est d'étudier la dette moyenne.

```
credit = pd.read_csv("tp3/Credit.csv")
```

Etude rapide du jeu de données

Les trois graphiques ci-dessous portent sur la cote de solvabilité.

- De quoi sont composés ces graphiques ?
- Comment les interprétez-vous ?

```
sns.lmplot(x='Rating', y='Limit', data=credit, lowess=True, scatter_kws={'s': 10}, line_kws={'color': 'red'})
plt.show()

sns.lmplot(x='Income', y='Rating', data=credit, lowess=True, scatter_kws={'s': 10}, line_kws={'color': 'red'})
plt.show()

sns.lmplot(x='Rating', y='Balance', data=credit, lowess=True, scatter_kws={'s': 10}, line_kws={'color': 'red'})
plt.show()
```

Il est facile de voir que quelques variables sont corrélées avec la dette moyenne. Voici trois graphiques qui montrent ces corrélations.

- Comment sont-ils construits ?
- Comment les interprétez-vous ?

```
sns.kdeplot(data=credit, x='Balance', hue='Student', fill=True, alpha=0.5, common_norm=False)
plt.show()

credit['Cards'] = credit['Cards'].astype('category')
sns.boxplot(x='Cards', y='Balance', data=credit)
credit['Cards'] = credit['Cards'].astype('float')

plt.show()
```

À l'aide de la commande `smf.ols`, ajuster le “modèle complet”, qui prédit la variable **Balance** avec l'ensemble de toutes les autres covariables. Enregistrer le résultat dans un objet nommé `m_complet`. Puis utiliser la méthode `.summary()` et un `print` pour obtenir des résultats numériques. Et, enfin, commentez les.

Le jeu de données contient des variables catégorielles. La matrice \mathbf{X} utilisée pour faire le calcul de $\hat{\beta}$ contient donc des variables binaires pour remplacer ces variables catégorielles.

- À la vue de ces sorties, quelles sont les covariables binaires qui ont été créées ? Quelles sont les modalités de référence qui ont été choisies ?
- La commande `m_complet.model.exog` permet de récupérer la matrice \mathbf{X} . Enregistrer dans un objet Python \mathbf{X} la valeur de `m_complet.model.exog`.
- Enregistrer dans un objet \mathbf{y} la valeur de `m_complet.model.endog`.
- À l'aide de la méthode `fit_transform` d'un `StandardScaler()`, enregistrer dans $\mathbf{X}_{\text{scaled}}$ la matrice des covariables centrées-réduites (après avoir enlevé la 1ère colonne de \mathbf{X} , qui est le vecteur constant 1.)
- Calculer la matrice de corrélation, et enregistrer la dans l'objet \mathbf{C} en utilisant la fonction `np.corrcoef` avec l'option `rowvar=False`.
- Calculer les valeurs propres de cette matrice avec la fonction `np.linalg.eigh` en utilisant l'option `envals_only=True`. Enregistrer le résultat dans un objet nommé `eigenvalues`.
- Afficher ces valeurs propres et conclure.

Ridge, avec validation croisée

Que fait le code ci-dessous ? (Noter que les différentes valeurs de λ testées sont choisies de façon régulière sur une échelle logarithmique.)

```
# Ajustements
lambda_grille = np.logspace(5, -3, 100)
ridge_cv = RidgeCV(alphas=lambda_grille, store_cv_values=True)
ridge_cv.fit(X_scaled, y) # Exclude the intercept column

# Erreur de validation croisée
cv_errors = np.mean(ridge_cv.cv_values_, axis=0)
cv_se = np.std(ridge_cv.cv_values_, axis=0) / np.sqrt(X_scaled.shape[0])

# Sélection de lambda avec la règle du min et du 1se
min_error_idx = np.argmin(cv_errors)
min_error = cv_errors[min_error_idx]
one_se_threshold = min_error + cv_se[min_error_idx]
lambda_min = lambda_grille[min_error_idx]
lambda_1se_idx = np.where(cv_errors <= one_se_threshold)[0][0]
lambda_1se = lambda_grille[lambda_1se_idx]
```

Remarque : on a utilisé deux façons de choisir λ par validation croisée :

- $\hat{\lambda}_{\min}$: la valeur de λ qui rend l'erreur de prédiction estimée par validation croisée la plus faible,
- $\hat{\lambda}_{1se}$: la valeur de λ la plus grande possible telle que l'erreur de prédiction moins l'erreur-type de cette estimation soit inférieure à l'erreur la plus faible.

Voici une représentation graphique de ces erreurs de validation croisée.

```
# Représentation des erreurs de validation croisée
erreur = pd.DataFrame({'Lambda': lambda_grille, 'CV error': cv_errors, 'CV se': cv_se})
plt.errorbar(erreur['Lambda'], erreur['CV error'], yerr=erreur['CV se'], fmt='o', \
             ecolor='gray', capsizes=3, label='CV error (+/- se)')
plt.axhline(y=np.var(y), color='green', linestyle='--', label='Var(y)')
plt.xscale('log')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
# Zoom sur la zone intéressante
plt.errorbar(erreur['Lambda'], erreur['CV error'], yerr=erreur['CV se'], fmt='o', \
             ecolor='gray', capsizes=3, label='CV error (+/- se)')
plt.axvline(x=lambda_min, linestyle='--', color='red', label='lambda_min')
plt.axvline(x=lambda_1se, linestyle='-.', color='red', label='lambda_1se')
plt.ylim(9400, 11000)
plt.xscale('log')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

- Relancer le bloc qui contient l'estimation avec `RidgeCV` et constater que l'estimation $\hat{\lambda}_{\min}$ est instable. (Vous pouvez jouer sur les valeurs des bornes de `ylim` pour zoomer sur certaines parties du graphique)

On peut ensuite représenter les différentes estimations $\hat{\beta}_\lambda$ de β en fonction de λ . Le code ci-dessous permet d'obtenir une figure satisfaisante.

```
coefs = []
intercepts = []
for alpha in lambda_grille:
    ridge = RidgeCV(alphas=[alpha])
    ridge.fit(X_scaled, y)
    coefs.append(ridge.coef_)
    intercepts.append(ridge.intercept_)

coefs = np.array(coefs)
intercepts = np.array(intercepts)
var_names = m_complet.model.exog_names

plt.figure(figsize=(10, 6))
colors = plt.cm.tab20(np.linspace(0, 1, coefs.shape[1]))
```

```

for i in range(coefs.shape[1]):
    plt.plot(lambda_grille, coefs[:, i], label=var_names[i+1], color=colors[i])

plt.axvline(x=lambda_min, linestyle='--', color='grey')
plt.text(lambda_min, plt.ylim()[0], 'lambda_min', rotation=90, verticalalignment='bottom', color='grey')

plt.axvline(x=lambda_1se, linestyle='-.', color='black')
plt.text(lambda_1se, plt.ylim()[0], 'lambda_1se', rotation=90, verticalalignment='bottom', color='black')

plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Coefficients')
plt.title('Evolution des coefficients en fonction de lambda')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()

```

Ce graphique représente l'évolution des différentes coordonnées (couleurs de courbes) de $\hat{\beta}_\lambda$ en fonction de λ (sur l'axe des abscisses).

- Interpréter ce graphique. Que se passe-t-il en $\lambda = \hat{\lambda}_{1se}$?
- Que peut-on dire sur les valeurs des effets estimés des variables `Limit` et `Rating` pour $\lambda = \hat{\lambda}_{1se}$? Que pouvait-on dire de ces variables à la vue des premiers graphiques de cet exercice ?

On peut récupérer les valeurs de $\hat{\beta}$ pour les deux choix de λ (min et 1se) avec le code ci-dessous.

```

res = pd.DataFrame({'Variable': ['Intercept'] + var_names[1:], \
                    'Estimates (lambda_min)': [intercepts[min_error_idx]] + list(coefs[min_error_idx]), \
                    'Estimates (lambda_1se)': [intercepts[lambda_1se_idx]] + list(coefs[lambda_1se_idx])})
print(f"Coefficients for lambda_min ({lambda_min}) and lambda_1se ({lambda_1se}):")
print(res)

```

Exercice 2 : Lasso et sélection de variables

Dans cet exercice, on s'intéresse à prédire le salaire de joueurs professionnels de la ligue majeure de Baseball lors des saisons de 1986 et 1987. Pour prédire ce salaire, on dispose de 19 covariables :

- `AtBat` : nombre de fois à la batte en 1986
- `Hits` : nombre de coups réussis en 1986
- `HmRun` : nombre de home runs en 1986
- `Runs` : nombre de runs en 1986
- `RBI` : nombre de points produits (puissance) en 1986
- `Walks` : nombre de buts sur balle en 1986

- **Years** : nombre d'année d'expérience dans la ligue majeure
- **CAtBat,..,CWalks** : même chose que précédemment, mais durant toute la carrière
- **League** : ligue du joueur à la fin de 1986
- **Division** : division du joueur à la fin de 1986
- **PutOuts** : nombre de retraits en 1986
- **Assists** : nombre d'assistance en 1986
- **Errors** : nombre d'erreurs en 1986
- **Salary** : salaire annuel au début de 1987, en milliers de dollars
- **NewLeague** : ligue du joueur au début de 1987.

Ce jeu de données se charge ainsi.

```
hitters = pd.read_csv("tp3/Hitters.csv", sep=",")
hitters.rename(columns=[hitters.columns[0]: 'Player'], inplace=True)
hitters['League'] = hitters['League'].astype('category')
hitters['Division'] = hitters['Division'].astype('category')
hitters['NewLeague'] = hitters['NewLeague'].astype('category')
hitters.dropna(inplace=True)
print(hitters.head())
```

En vous inspirant de ce qui a été fait dans le 1er exercice :

- Ajuster le modèle complet qui prédit le salaire en fonction de toutes les autres covariables et étudier les résultats numériques obtenus.
- Etudier la matrice de corrélation de \mathbf{X} et ses valeurs propres.
- Conclure.

Lasso

On peut obtenir les estimations Lasso de β pour différentes valeurs de λ ainsi.

```
# Calcul de X_scaled et y
covariates = hitters.columns.copy()
covariates = covariates.drop(['Player', 'Salary'])
formula = 'Salary ~ ' + ' + '.join(covariates)
m_complet = smf.ols(formula=formula, data=hitters).fit()

X = m_complet.model.exog
y = m_complet.model.endog

X_scaled = StandardScaler().fit_transform(X[:, 1:])
```

```

# Lasso
# Ajustements
lambda_grille = np.logspace(3, -2, 100)
lasso_cv = LassoCV(alphas=lambda_grille, cv=10)
lasso_cv.fit(X_scaled, y) # Exclude the intercept column

# Erreur de validation croisée
cv_errors = np.mean(lasso_cv.mse_path_, axis=1)
cv_se = np.std(lasso_cv.mse_path_, axis=1) / np.sqrt(10)

# Sélection de lambda avec la règle du min et du 1se
min_error_idx = np.argmin(cv_errors)
min_error = cv_errors[min_error_idx]
one_se_threshold = min_error + cv_se[min_error_idx]
lambda_min = lambda_grille[min_error_idx]
lambda_1se_idx = np.where(cv_errors <= one_se_threshold)[0][0]
lambda_1se = lambda_grille[lambda_1se_idx]

```

Commenter les résultats du graphique ci-dessous.

```

erreur = pd.DataFrame({'Lambda': lambda_grille, 'CV error': cv_errors, 'CV se': cv_se})
plt.errorbar(erreur['Lambda'], erreur['CV error'], yerr=erreur['CV se'], fmt='o', \
             ecolor='gray', capsizes=3, label='CV error (+/- se)')
plt.axhline(y=np.var(y), color='green', linestyle='--', label='Var(y)')
plt.xscale('log')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
# Zoom sur la zone intéressante
plt.errorbar(erreur['Lambda'], erreur['CV error'], yerr=erreur['CV se'], fmt='o', \
             ecolor='gray', capsizes=3, label='CV error (+/- se)')
plt.axvline(x=lambda_min, linestyle='--', color='red', label='lambda_min')
plt.axvline(x=lambda_1se, linestyle='-.', color='red', label='lambda_1se')
plt.ylim(80000, 160000)
plt.xscale('log')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()

```

Comme pour la régression ridge, on peut représenter l'évolution de $\hat{\beta}_\lambda$ en fonction de λ . Voici le code qui permet de le faire.

```

coefs = []
intercepts = []
for alpha in lambda_grille:

```

```

lasso = Lasso(alpha=alpha, tol=1e-5, max_iter=10000)
lasso.fit(X_scaled, y)
coefs.append(lasso.coef_)
intercepts.append(lasso.intercept_)

coefs = np.array(coefs)
intercepts = np.array(intercepts)
var_names = m_complet.model.exog_names

plt.figure(figsize=(10, 6))
colors = plt.cm.tab20(np.linspace(0, 1, coefs.shape[1]))
for i in range(coefs.shape[1]):
    plt.plot(lambda_grille, coefs[:, i], label=var_names[i+1], color=colors[i])

plt.axvline(x=lambda_min, linestyle='--', color='grey')
plt.text(lambda_min, plt.ylim()[0], 'lambda_min', rotation=90, verticalalignment='bottom', color='grey')

plt.axvline(x=lambda_1se, linestyle='-.', color='black')
plt.text(lambda_1se, plt.ylim()[0], 'lambda_1se', rotation=90, verticalalignment='bottom', color='black')

plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Coefficients')
plt.title('Evolution des coefficients en fonction de lambda')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()

```

- Peut-on voir sur ce graphique que la méthode Lasso annule certaines coordonnées de β ? Comparer avec le même type graphique pour la méthode ridge dans l'exercice 1.
- Récupérer l'estimation de β pour les deux choix de λ (min et 1se), et constater que certaines coordonnées ont été annulées.

```

res = pd.DataFrame({'Variable': ['Intercept'] + var_names[1:], \
                    'Estimates (lambda_min)': [intercepts[min_error_idx]] + list(coefs[min_error_idx]), \
                    'Estimates (lambda_1se)': [intercepts[lambda_1se_idx]] + list(coefs[lambda_1se_idx])})
print(f"Coefficients for lambda_min ({lambda_min}) and lambda_1se ({lambda_1se}):")
print(res)

```

Sélection de covariables

On commence par la méthode progressive.

- On utilise ici le `ForwardModelSelector` programmé dans `ModelSelector.py` pour mettre en œuvre la méthode de sélection progressive.
- Quel est le modèle choisi par le critère AIC ? Par le critère BIC ?

```
from ModelSelector import ForwardModelSelector
selector = ForwardModelSelector(formula=formula, dataset=hitters)
selector = selector.fit()
bic_model = selector.best_model('BIC').fit()
aic_model = selector.best_model('AIC').fit()
print(bic_model.summary())
print(aic_model.summary())
```

Même étude avec la méthode rétrograde

- Implémenter la méthode rétrograde.
- Quelles sont les deux premières variables à sortir du modèle complet ?
- Quels sont les modèles choisis par les critères BIC, AIC ?
- Comparer les résultats obtenus entre les méthodes progressives et rétrogrades. Comparer également les résultats avec ceux de l'estimation Lasso.