Apprentissage automatique Chapitre 2 - Apprentissage supervisé et non supervisé

Halim Djerroud

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

révision : 0.1

Plan du cours et déroulement

Plan du cours

1. Introduction.
2. Les Données.
3. **Apprentissage supervisé et non supervisé.**
4. Les réseaux de neurones.

Déroulement

- 18 heures de cours 6 séances de 3 heures.
- Deux contrôles continus (QCM).
- Un projet à faire en binôme.
- Un examen écrit.

Approche du machine learning

Apprentissage supervisé :

- Régression linéaire simple
- Régression linéaire multiple
- Équation normale
- Régression polynomiale
- Modèles linéaires régularisés
- Réseau de neurone
- Machine à vecteurs de support linéaire et non linéaire (SVM)

Apprentissage non supervisé :

- K-NN, K-MEANS
- Arbre de décision

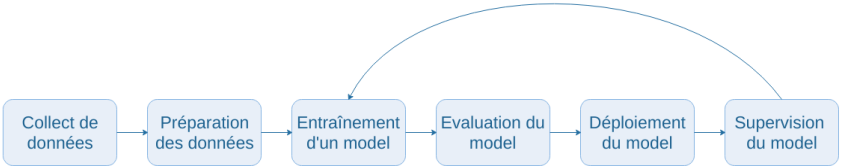
Analyse des séries chronologiques :

- Modèle ARIMA

Apprentissage par renforcement :

- Processus de décision markovien

Quelle est l'approche machine learning ?



1. Analyser les données,
2. Choisir un modèle,
3. Les modèles sont entraînés avec des données (data mining),
4. Estimer l'erreur du modèle,
5. Mettre à jour le modèle.

Contraintes:

- Les données doivent être de très bonne qualité
- Le volume des données est important pour entraîner le modèle

Les types d'apprentissage

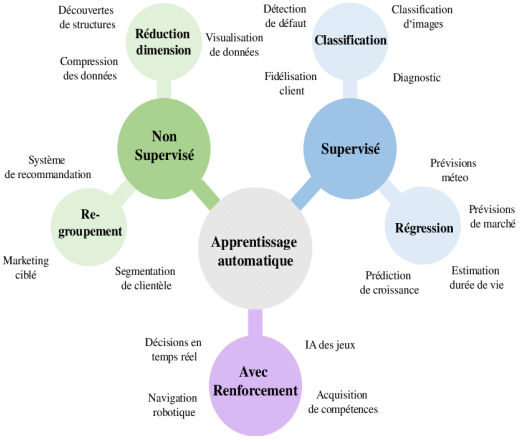


Figure: Source web

Apprentissage supervisé

Apprentissage supervisé : Régression linéaire

Objectifs du chapitre :

1. Régression linéaire

Régression linéaire

Définition

Une régression linéaire est un modèle qui cherche à établir une relation linéaire entre une variable, dite expliquée, et une ou plusieurs variables, dites explicatives.

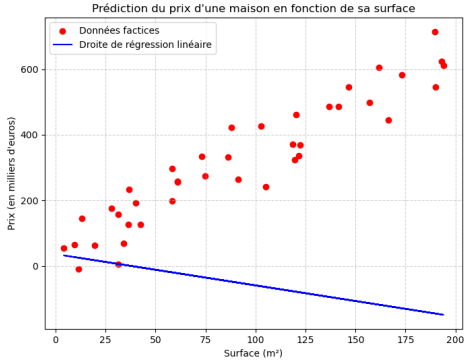
- La régression linéaire est une méthode statistique utilisée pour modéliser la relation entre une variable dépendante (variable cible) et une ou plusieurs variables indépendantes (prédicteurs).
- L'objectif est de trouver la meilleure droite qui prédit la variable dépendante en fonction des variables indépendantes.
- Permet de modéliser la relation entre une variable dépendante Y et une variable indépendante X.
- L'équation du modèle est : $Y = aX + b$
- Les paramètres a (pente) et b (ordonnée à l'origine) sont estimés par *la méthode des moindres carrés*.

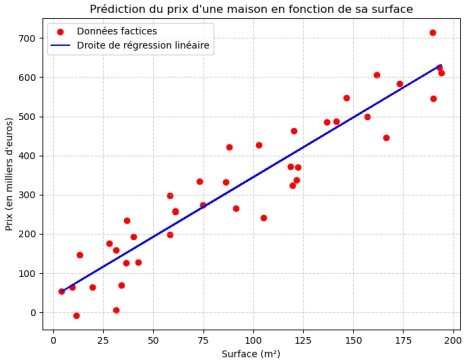
Hypothèses principales

- **Linarité** : La relation entre la variable dépendante et les variables indépendantes est linéaire.
- **Indépendance des erreurs** : Les résidus doivent être indépendants.
- **Homoscédasticité** : La variance des erreurs est constante pour toutes les valeurs des variables indépendantes.
- **Normalité des erreurs** : Les erreurs suivent une distribution normale.
- **Absence de multicollinéarité** : Les variables indépendantes ne doivent pas être trop fortement corrélées entre elles.

Exemple

Prédire le prix d'une maison en fonction de sa surface.





Le but est d'ajuster les coefficients a et b pour aligner la droite le mieux possible.

LISV Laboratoire d'Ingénierie des systèmes de Versailles — UVSQ université PARIS-SACLAY

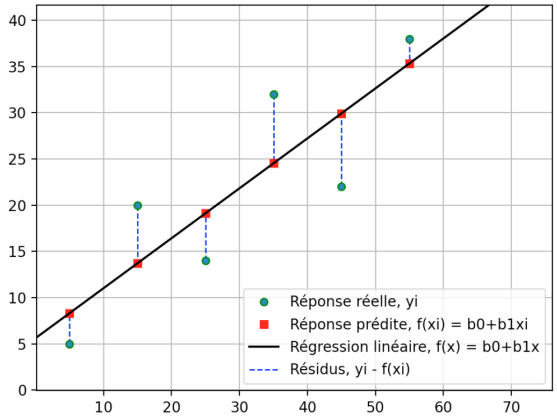
Formulation mathématique

- **Forme générale du modèle** :

$$y = \beta_0 + \beta_1 x + \varepsilon \tag{1}$$

- y : variable dépendante (prédiction)
- x : variable indépendante (prédicteur)
- β_0 : intercept (ordonnée à l'origine)
- β_1 : pente de la droite
- ε : erreur aléatoire

Représentation graphique



- La droite de régression minimise la somme des carrés des résidus.

Méthode des moindres carrés

- La méthode des moindres carrés consiste à minimiser la somme des carrés des résidus $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
- Les formules pour les coefficients sont :

$$a = \frac{COV(X;Y)}{V(X)} = \frac{n\sum(x_i y_i) - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$$

$$b = \bar{Y} - a\bar{X} = \frac{\sum y_i - a\sum x_i}{n}$$

$$y = aX + b$$

Problèmes avec la méthode des moindres carrés

- **Sensibilité aux valeurs aberrantes** : Les valeurs extrêmes peuvent avoir un impact significatif sur la droite de régression.
- **Supposition de linéarité** : La méthode suppose que la relation entre les variables est linéaire.
- **Multicolinéarité** : La présence d'une forte corrélation entre les variables indépendantes peut rendre l'estimation des coefficients instable.
- **Hétéroscédasticité** : Lorsque la variance des erreurs n'est pas constante, les coefficients peuvent être biaisés.
- **Non-normalité des résidus** : Si les résidus ne sont pas normalement distribués, les tests statistiques basés sur cette hypothèse peuvent être invalides.
- **Inversion de matrice coûteuse** : La résolution du système des équations normales implique l'inversion d'une matrice, ce qui est coûteux en termes de calcul, notamment pour les grands ensembles de données.

Mesures de performance : il existe plusieurs

- Erreur quadratique moyenne (MSE) :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

où n est le nombre total d'observations.

- Racine de l'erreur quadratique moyenne (RMSE) :

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Le RMSE est la racine carrée du MSE et a la même unité que la variable cible y.

- Erreur absolue moyenne (MAE) :

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

MAE mesure la moyenne des écarts absolus entre les valeurs prédites et les valeurs réelles.

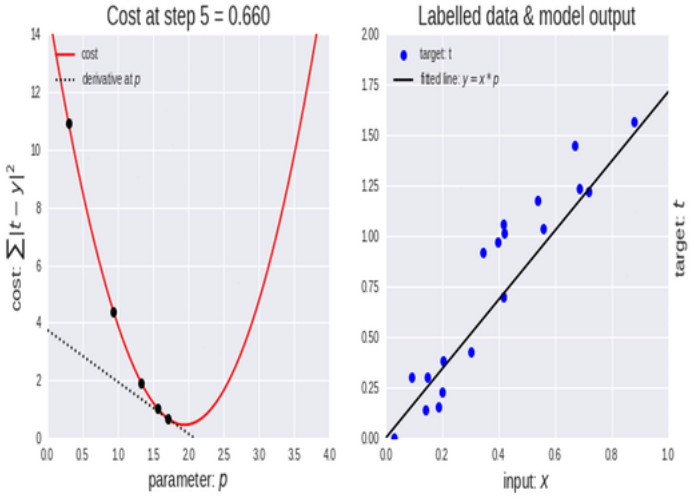
Principe de la descente de gradient

- La descente de gradient est une méthode d'optimisation itérative pour minimiser une fonction coût.
- L'objectif est de mettre à jour les paramètres β_0 et β_1 de manière itérative :

$$\beta_j := \beta_j - a\frac{\partial J}{\partial \beta_j} \tag{2}$$

- a est le taux d'apprentissage, J est la fonction coût.

Algorithme de la descente de gradient



Algorithme de la descente de gradient

1. Initialiser β_0 et β_1 avec des valeurs aléatoires.
2. Répéter jusqu'à convergence :
   - Calculer le gradient ∇J.
   - Mettre à jour les paramètres $\beta_j := \beta_j - a\nabla J$.

Descente de Gradient et RMSE

- La descente de gradient est une méthode itérative pour optimiser la fonction de coût.
- La fonction de coût utilisée pour la régression linéaire est l'erreur quadratique moyenne (MSE) :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3}$$

- La racine carrée de cette valeur est le RMSE (Root Mean Squared Error) :

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4}$$

- L'objectif de la descente de gradient est de minimiser le RMSE.

Méthodes de validation

- Diviser les données en ensembles d'entraînement et de test.
- Calculer le coefficient de détermination R^2 sur les données de test.
- Analyser les résidus pour évaluer la qualité de la régression.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

où y_i sont les valeurs réelles, $\hat{y}_i$ les valeurs prédites, et $\bar{y}$ la moyenne des y.

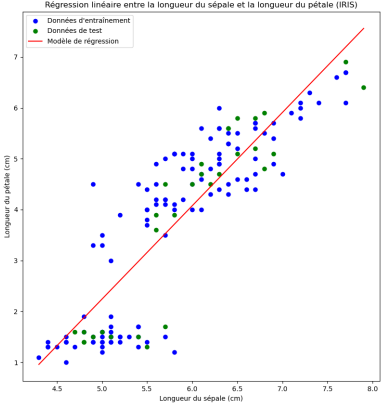
Exemple

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_iris
# Chargement du dataset IRIS
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
# Selection des variables (longueur du sepale et longueur du petale)
X = df[['sepal length (cm)']]
y = df['petal length (cm)']
# Separation des donnees en ensemble d'entrainement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Creation et entrainement du modele de regression lineaire
model = LinearRegression()
model.fit(X_train, y_train)
# Prediction sur les donnees de test
y_pred = model.predict(X_test)
```

Exemple

```
# Calcul des metriques de performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Erreur quadratique moyenne (MSE) : {mse:.2f}')
print(f'Coefficient de determination (R²) : {r2:.2f}')
# Validation croisee
scores = cross_val_score(model, X, y, cv=5, scoring='r2')
print(f'Scores R² de validation croisee : {scores}')
print(f'Moyenne R² : {scores.mean():.2f}')
# Visualisation des resultats
plt.scatter(X_train, y_train, color='blue', label='Donnees d\'entrainement')
plt.scatter(X_test, y_test, color='green', label='Donnees de test')
# Affichage de la droite de regression lisse
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
plt.plot(X_range, model.predict(X_range), color='red', label='Modele de regression')
plt.xlabel('Longueur du sepale (cm)')
plt.ylabel('Longueur du petale (cm)')
plt.title('Regression lineaire entre la longueur
          du sepale et la longueur du petale (IRIS)')
plt.legend()
plt.show()
```

Exemple




```
Erreur quadratique moyenne (MSE) : 0.60
Coefficient de determination (R²) : 0.82
Scores R² de validation croisee :
[-51.52454692   0.52866261  -1.85363774
 -0.10127352  -1.87189524]
Moyenne R² : -10.96
```

LISV Laboratoire d'Ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Exemple avec uniquement "Iris Virginica"

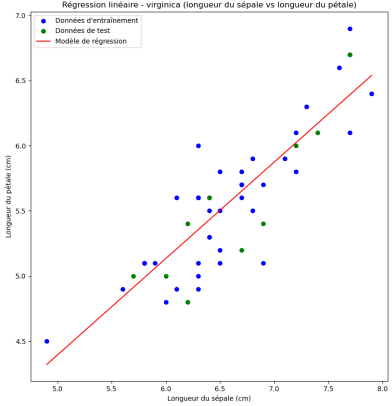
```
...
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target  # Ajout de la colonne cible (0, 1 ou 2)

# Filtrer le DataFrame pour ne conserver qu'une seule
# classe (exemple : classe 0 - Setosa)

classe_cible = 2  # Remplacez par 1 (Versicolor)
                  # ou 2 (Virginica) pour changer de classe
df_classe = df[df['target'] == classe_cible]

# Selection des variables (longueur du sepale et longueur du petale)
# pour la classe filtree
X = df_classe[['sepal length (cm)']]
y = df_classe['petal length (cm)']
...
```

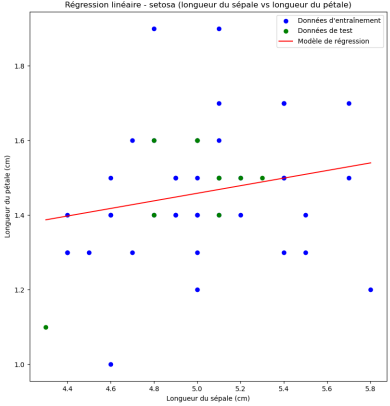
Exemple avec la classe virginica




```
Classe selectionnee : virginica
Erreur quadratique moyenne (MSE) : 0.08
Coefficient de determination (R²) : 0.76
Scores R² de validation croisee : [ 0.64361626
  0.8314034   0.78018601  0.61811866 -0.21852765]
Moyenne R² : 0.53
```

LISV Laboratoire d'Ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Exemple avec la classe versicolor




```
Classe selectionnee : versicolor
Erreur quadratique moyenne (MSE) : 0.10
Coefficient de determination (R²) : 0.59
Scores R² de validation croisee : [0.79778748
0.4277642  0.30493865 0.28112056 0.54946235]
```

LISV Laboratoire d'Ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

Apprentissage supervisé

Apprentissage supervisé : Régression linéaire multiple

Objectifs du chapitre :

1. Comprendre la régression linéaire multiple
2. Formuler le modèle en notation matricielle
3. Maîtriser l'équation normale
4. Détecter et gérer la multicolinéarité
5. Sélectionner les variables pertinentes

Rappel : Régression linéaire simple

Modèle simple :

- Une seule variable explicative
- Équation : $y = \beta_0 + \beta_1 x + \varepsilon$
- Exemple : Prix d'une maison en fonction de sa surface

Limites :

- La réalité est rarement univariée
- Plusieurs facteurs influencent souvent le résultat
- Besoin de modèles plus complexes

Régression linéaire multiple

Définition

La régression linéaire multiple modélise la relation entre une variable dépendante Y et plusieurs variables indépendantes $X_1, X_2, \ldots, X_p$.

- **Forme générale :**

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \varepsilon \tag{5}$$

- y : variable dépendante (cible)
- $x_1, x_2, \ldots, x_p$: variables indépendantes (prédicteurs)
- β_0 : intercept (ordonnée à l'origine)
- $\beta_1, \ldots, \beta_p$: coefficients de régression
- ε : erreur aléatoire

Exemple concret

Prédire le prix d'une maison avec plusieurs caractéristiques :

- **Variable dépendante** : Prix de la maison (en milliers d'euros)
- **Variables indépendantes** :
  - x_1 : Surface habitable (m^2)
  - x_2 : Nombre de chambres
  - x_3 : Âge de la maison (années)
  - x_4 : Distance au centre-ville (km)
  - x_5 : Surface du jardin (m^2)

Modèle :

$$\text{Prix} = \beta_0 + \beta_1 \times \text{Surface} + \beta_2 \times \text{Chambres} + \ldots + \varepsilon$$

Notation matricielle

Pour n observations et p variables :

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon \tag{6}$$

Où :

- $\mathbf{Y}$: vecteur des observations $(n \times 1)$

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

- $\mathbf{X}$: matrice de design $(n \times (p+1))$

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & & & & \end{pmatrix}$$

Notation matricielle (suite)

- β : vecteur des coefficients $((p+1) \times 1)$

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}$$

- ε : vecteur des erreurs $(n \times 1)$

$$\varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

Avantage : Notation compacte et généralisation à p variables

Estimation des paramètres : Équation normale

Objectif : Minimiser la somme des carrés des résidus (SSR)

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = ||\mathbf{Y} - \mathbf{X}\beta||^2 \tag{7}$$

Solution analytique (Équation normale) :

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{8}$$

- Solution exacte (pas d'itération nécessaire)
- Nécessite l'inversion de $\mathbf{X}^T\mathbf{X}$
- Complexité : $O(p^3)$ pour l'inversion
- Problème si $\mathbf{X}^T\mathbf{X}$ n'est pas inversible

Équation normale : Démonstration

On cherche à minimiser :

$$J(\beta) = ||\mathbf{Y} - \mathbf{X}\beta||^2 = (\mathbf{Y} - \mathbf{X}\beta)^T(\mathbf{Y} - \mathbf{X}\beta)$$

Développement :

$$\begin{aligned} J(\beta) &= \mathbf{Y}^T\mathbf{Y} - \mathbf{Y}^T\mathbf{X}\beta - \beta^T\mathbf{X}^T\mathbf{Y} + \beta^T\mathbf{X}^T\mathbf{X}\beta \\ &= \mathbf{Y}^T\mathbf{Y} - 2\beta^T\mathbf{X}^T\mathbf{Y} + \beta^T\mathbf{X}^T\mathbf{X}\beta \end{aligned}$$

Dérivée par rapport à β :

$$\frac{\partial J}{\partial \beta} = -2\mathbf{X}^T\mathbf{Y} + 2\mathbf{X}^T\mathbf{X}\beta = 0$$

D'où : $\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{Y}$

Interprétation des coefficients

- β_0 : Valeur prédite de Y quand toutes les variables $X_i = 0$
- β_i : Variation moyenne de Y pour une augmentation d'une unité de X_i, **toutes autres variables constantes**
- **Exemple (prix de maison) :**
  - Si $\beta_1 = 2.5$ (coefficient de la surface)
  - Pour chaque m^2 supplémentaire, le prix augmente de 2500€ en moyenne
  - **À condition que** : nombre de chambres, âge, distance, etc. restent constants
- **Attention :** L'interprétation suppose l'absence de multicolinéarité

Multicolinéarité

Définition

La multicolinéarité existe quand deux ou plusieurs variables indépendantes sont fortement corrélées entre elles.

Conséquences :

- Les coefficients β_i deviennent instables
- Grande variance des estimateurs
- Difficulté à isoler l'effet de chaque variable
- $\mathbf{X}^T\mathbf{X}$ devient singulière ou presque singulière
- Inversion numérique instable

Exemple :

- Surface habitable et nombre de pièces souvent corrélés
- Âge de la maison et état de la toiture corrélés

Détection de la multicolinéarité

1. Matrice de corrélation

- Calculer les corrélations deux à deux entre variables
- Seuil : $|r| > 0.8$ ou 0.9 indique un problème

2. Facteur d'inflation de la variance (VIF)

$$VIF_i = \frac{1}{1 - R_i^2} \tag{9}$$

- R_i^2 : coefficient de détermination de la régression de X_i sur les autres variables
- VIF > 10 : multicolinéarité préoccupante
- VIF > 5 : multicolinéarité modérée

3. Condition Number

- Basé sur les valeurs propres de $\mathbf{X}^T\mathbf{X}$
- CN > 30 indique un problème

Solutions à la multicolinéarité

1. Éliminer des variables

- Supprimer une des variables corrélées
- Choisir la moins pertinente théoriquement

2. Combiner des variables

- Créer une variable composite (ex: moyenne, somme)
- Analyse en composantes principales (PCA)

3. Régularisation

- Ridge Regression (L2)
- Lasso Regression (L1)
- Elastic Net

4. Collecter plus de données

- Augmenter la taille de l'échantillon

Sélection de variables

Pourquoi sélectionner ?

- Éviter le surapprentissage
- Améliorer l'interprétabilité
- Réduire le coût de calcul
- Éliminer les variables non pertinentes

Méthodes principales :

1. **Forward Selection** : Ajout progressif de variables
2. **Backward Elimination** : Suppression progressive de variables
3. **Stepwise Selection** : Combinaison des deux précédentes
4. **Régularisation** : Lasso (met certains β_l à zéro)
5. **Critères statistiques** : AIC, BIC, R^2 ajusté

Critères de sélection

1. Critère d'Akaike (AIC)

$$AIC = 2k - 2\ln(\hat{L}) \tag{10}$$

- k : nombre de paramètres
- $\hat{L}$: vraisemblance maximale
- Plus l'AIC est faible, meilleur est le modèle

2. Critère Bayésien (BIC)

$$BIC = k\ln(n) - 2\ln(\hat{L}) \tag{11}$$

- Pénalise plus fortement la complexité que l'AIC

3. R^2 ajusté

$$R^2_{adj} = 1 - \frac{(1 - R^2)(n-1)}{n-p-1} \tag{12}$$

- Pénalise l'ajout de variables non pertinentes

Validation du modèle

Hypothèses à vérifier :

1. **Linéarité** : Relation linéaire entre Y et les X_i
   - Graphique : Résidus vs. valeurs prédites
2. **Indépendance des erreurs** : Pas d'autocorrélation
   - Test de Durbin-Watson
3. **Homoscédasticité** : Variance constante des erreurs
   - Test de Breusch-Pagan
4. **Normalité des résidus** : Distribution normale
   - Q-Q plot, test de Shapiro-Wilk
5. **Absence de multicolinéarité** : VIF < 10

Mesures de performance

1. Coefficient de détermination

$$R^2 = 1 - \frac{SSR}{SST} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{13}$$

- Proportion de variance expliquée
- $0 \leq R^2 \leq 1$ (plus proche de 1 = meilleur)

2. Erreurs de prédiction

- MSE : $\frac{1}{n}\sum(y_i - \hat{y}_i)^2$
- RMSE : $\sqrt{MSE}$
- MAE : $\frac{1}{n}\sum|y_i - \hat{y}_i|$

3. Tests de significativité

- Test de Fisher (significativité globale)
- Test de Student (significativité individuelle des β_i)

Comparaison : Simple vs. Multiple

Régression simple

- Une variable X
- Équation : $y = \beta_0 + \beta_1 x$
- Visualisation 2D facile
- Interprétation directe
- Peu de risque de multicolinéarité

Régression multiple

- Plusieurs variables X_i
- Équation matricielle
- Visualisation complexe
- Interprétation "toutes choses égales"
- Risque de multicolinéarité
- Plus réaliste

Quand utiliser la régression multiple ?

- Phénomène complexe avec plusieurs causes
- Besoin de contrôler des variables confondantes
- Amélioration significative du R^2

Exemple Python : Chargement des données

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.datasets import fetch_california_housing

# Chargement du dataset California Housing
housing = fetch_california_housing()
df = pd.DataFrame(data=housing.data, columns=housing.feature_names)
df['Price'] = housing.target  # Prix en 100k$

# Affichage des premières lignes
print(df.head())
print(f"\nDimensions : {df.shape}")
print(f"\nVariables : {list(df.columns)}")

# Matrice de correlation
corr_matrix = df.corr()
print(f"\nCorrelation avec le prix :\n{corr_matrix['Price'].sort_values(ascending=False)}")
```

Exemple Python : Détection de la multicolinéarité

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Selection de quelques variables pour l'exemple
features = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population']
X = df[features]
y = df['Price']

# Calcul du VIF pour chaque variable
vif_data = pd.DataFrame()
vif_data["Variable"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                   for i in range(len(X.columns))]

print("Facteur d'Inflation de la Variance (VIF) :")
print(vif_data)
print("\nInterpretation :")
print("VIF < 5 : Pas de multicolinearite")
print("5 < VIF < 10 : Multicolinearite moderee")
print("VIF > 10 : Multicolinearite sevEre")
```

Exemple Python : Entraînement du modèle

```python
# Separation des donnees
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Creation et entraînement du modèle
model = LinearRegression()
model.fit(X_train, y_train)

# Affichage des coefficients
print("Coefficients du modèle :")
print(f"Intercept (B) : {model.intercept_:.4f}")
for feature, coef in zip(features, model.coef_):
    print(f"B_{feature} : {coef:.4f}")

# Predictions
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

print("\nInterpretation exemple :")
print("Si MedInc augmente de 1 (10k$), le prix augmente de",
      f"{model.coef_[0]:.4f} × 100k$ = {model.coef_[0]*100:.0f}k$")
```

Exemple Python : Évaluation des performances

```python
# Metriques sur l'ensemble d'entraînement
train_mse = mean_squared_error(y_train, y_pred_train)
train_rmse = np.sqrt(train_mse)
train_mae = mean_absolute_error(y_train, y_pred_train)
train_r2 = r2_score(y_train, y_pred_train)

# Metriques sur l'ensemble de test
test_mse = mean_squared_error(y_test, y_pred_test)
test_rmse = np.sqrt(test_mse)
test_mae = mean_absolute_error(y_test, y_pred_test)
test_r2 = r2_score(y_test, y_pred_test)

print("=== PERFORMANCES ===")
print(f"\nEntraînement :")
print(f"  R² : {train_r2:.4f}")
print(f"  RMSE : {train_rmse:.4f} (× 100k$)")
print(f"  MAE : {train_mae:.4f} (× 100k$)")

print(f"\nTest :")
print(f"  R2 : {test_r2:.4f}")
print(f"  RMSE : {test_rmse:.4f} (× 100k$)")
print(f"  MAE : {test_mae:.4f} (× 100k$)")
```

Exemple Python : Visualisation des résidus

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Graphique 1 : Valeurs predites vs. reelles
axes[0].scatter(y_test, y_pred_test, alpha=0.5)
axes[0].plot([y_test.min(), y_test.max()],
             [y_test.min(), y_test.max()], 'r--', lw=2)
axes[0].set_xlabel('Valeurs reelles')
axes[0].set_ylabel('Valeurs predites')
axes[0].set_title('Predictions vs. Valeurs reelles')

# Graphique 2 : Residus vs. valeurs predites
residuals = y_test - y_pred_test
axes[1].scatter(y_pred_test, residuals, alpha=0.5)
axes[1].axhline(y=0, color='r', linestyle='--', lw=2)
axes[1].set_xlabel('Valeurs predites')
axes[1].set_ylabel('Residus')
axes[1].set_title('Analyse des residus')

plt.tight_layout()
plt.show()
```

Exemple Python : Équation normale

```python
# Implementation manuelle de l'equation normale
# B = (X^T X)^(-1) X^T Y

# Ajouter une colonne de 1 pour l'intercept
X_train_with_intercept = np.column_stack([np.ones(len(X_train)), X_train])

# Calcul de l'equation normale
XtX = X_train_with_intercept.T @ X_train_with_intercept
XtY = X_train_with_intercept.T @ y_train
beta = np.linalg.inv(XtX) @ XtY

print("Coefficients calcules avec l'equation normale :")
print(f"B0 (intercept) : {beta[0]:.4f}")
for i, feature in enumerate(features):
    print(f"B_{feature} : {beta[i+1]:.4f}")

print("\nComparaison avec sklearn :")
print(f"Difference intercept : {abs(beta[0] - model.intercept_):.6f}")
print(f"Difference coefficients : {np.max(np.abs(beta[1:] - model.coef_)):.6f}")
```

Exercice pratique

Données : Dataset "Boston Housing" ou créez vos propres données

À faire :

1. Charger et explorer les données
2. Visualiser la matrice de corrélation
3. Calculer les VIF pour détecter la multicolinéarité
4. Entraîner un modèle de régression linéaire multiple
5. Évaluer les performances (R^2, RMSE, MAE)
6. Analyser les résidus
7. Comparer avec un modèle de régression simple
8. Sélectionner les variables les plus pertinentes

Bonus :

- Implémenter l'équation normale manuellement
- Comparer avec la descente de gradient
- Tester différentes combinaisons de variables

Résumé du chapitre

Concepts clés :

- Régression linéaire multiple
- Notation matricielle
- Équation normale
- Multicolinéarité (VIF)
- Sélection de variables
- Critères AIC, BIC, R^2 ajusté

Points importants :

- Plus réaliste que la régression simple
- Attention à la multicolinéarité
- Vérifier les hypothèses
- Interpréter "toutes choses égales"
- Privilégier la simplicité
- Valider sur données de test

Formule essentielle :

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Pour aller plus loin

Méthodes avancées :

- **Régularisation** : Ridge, Lasso, Elastic Net
- **Régression polynomiale** : Relations non linéaires
- **Interactions** : Produits entre variables
- **Transformations** : Log, racine carrée, Box-Cox
- **Variables catégorielles** : One-hot encoding
- **Régression robuste** : Insensible aux valeurs aberrantes

Prochaine étape :

- Classification (régression logistique)
- Arbres de décision
- Méthodes d'ensemble

Apprentissage supervisé

Apprentissage supervisé : Régression Logistique

Objectifs du chapitre :

1. Comprendre la classification binaire
2. Maîtriser la régression logistique
3. Fonction sigmoïde et interprétation probabiliste
4. Fonction de coût et optimisation
5. Métriques d'évaluation (accuracy, précision, recall, F1)
6. Matrice de confusion et courbe ROC

De la régression à la classification

Régression (vue précédemment) :

- Prédire une valeur continue
- Exemples : prix, température, âge
- Sortie : $y \in \mathbb{R}$

Régression linéaire :

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$$

Classification :

- Prédire une catégorie/classe
- Exemples : spam/non-spam, malade/sain
- Sortie : $y \in \{0, 1\}$ (binaire)

Régression logistique :

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + ... + \beta_p x_p)}}$$

Qu'est-ce que la classification ?

Définition

La classification consiste à prédire la classe ou catégorie d'une observation à partir de ses caractéristiques.

Types de classification :

- **Classification binaire** : 2 classes (0 ou 1)
  - Exemples : Email spam/non-spam, Tumeur bénigne/maligne, Crédit accordé/refusé
- **Classification multiclasse** : Plus de 2 classes
  - Exemples : Reconnaissance de chiffres (0-9), Types de fleurs (Iris)

Dans ce chapitre : Focus sur la classification binaire

Pourquoi pas la régression linéaire pour classifier ?

Problème :

- La régression linéaire prédit des valeurs continues
- Pour une classification : besoin de $\hat{y} \in [0, 1]$
- Régression linéaire peut donner $\hat{y} < 0$ ou $\hat{y} > 1$
- Pas d'interprétation probabiliste claire

Solution : Utiliser une fonction sigmoïde !

Exemple :

- Prédire si un étudiant réussit (1) ou échoue (0)
- Variable : heures d'étude
- Régression linéaire pourrait prédire 1.5 ou -0.3
- Besoin d'une fonction qui "écrase" les valeurs entre 0 et 1

La fonction sigmoïde (logistique)

Définition :

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad (14)$$

Propriétés :

- Sortie : $\sigma(z) \in (0, 1)$
- $\sigma(0) = 0.5$
- $\lim_{z \to +\infty} \sigma(z) = 1$
- $\lim_{z \to -\infty} \sigma(z) = 0$
- Fonction en forme de "S"
- Dérivée : $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

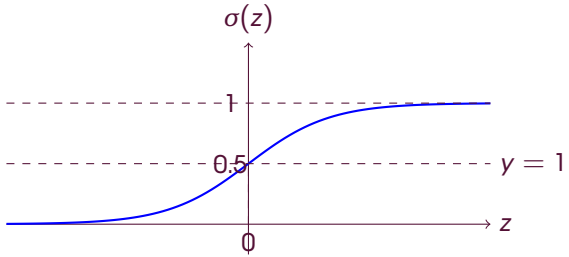


Figure: Courbe de la fonction sigmoïde

Modèle de régression logistique

Pour une observation avec caractéristiques $\mathbf{x} = (x_1, x_2, ..., x_p)$:

1. **Combinaison linéaire :**

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p = \beta^T \mathbf{x} \tag{15}$$

2. **Application de la sigmoïde :**

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} = P(y = 1|\mathbf{x}) \tag{16}$$

3. **Décision :**

$$\text{Classe prédite} = \begin{cases} 1 & \text{si } \hat{y} \geq 0.5 \\ 0 & \text{si } \hat{y} < 0.5 \end{cases} \tag{17}$$

Interprétation : $\hat{y}$ est la probabilité que l'observation appartienne à la classe 1

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Notation matricielle

Pour n observations :

- **Matrice de design X** : $(n \times (p+1))$
- **Vecteur des paramètres** β : $((p+1) \times 1)$
- **Vecteur des prédictions** $\hat{\mathbf{y}}$: $(n \times 1)$

Modèle complet :

$$\hat{\mathbf{y}} = \sigma(\mathbf{X}\beta) \tag{18}$$

où σ est appliquée élément par élément.

Note : Contrairement à la régression linéaire, il n'existe pas de solution analytique simple pour trouver β optimal.

Fonction de coût : Log-Loss (Entropie croisée)

Pourquoi pas MSE ? Le MSE n'est pas convexe pour la régression logistique.

Fonction de coût (Binary Cross-Entropy) :

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] \tag{19}$$

Intuition :

- Si $y_i = 1$: on veut $\hat{y}_i \to 1$ (minimiser $-\log(\hat{y}_i)$)
- Si $y_i = 0$: on veut $\hat{y}_i \to 0$ (minimiser $-\log(1-\hat{y}_i)$)
- Fonction convexe : garantit un minimum global
- Pénalise fortement les prédictions confiantes mais fausses

Cas particuliers :

- Si $y = 1$ et $\hat{y} = 1$: coût = 0
- Si $y = 1$ et $\hat{y} \to 0$: coût $\to +\infty$

Optimisation : Descente de gradient

Objectif : Minimiser $J(\beta)$

Gradient de la fonction de coût :

$$\frac{\partial J}{\partial \beta_j} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_{ij} \tag{20}$$

Forme matricielle :

$$\nabla J = \frac{1}{n}\mathbf{X}^T(\hat{\mathbf{y}} - \mathbf{y}) \tag{21}$$

Mise à jour des paramètres :

$$\beta := \beta - a\nabla J \tag{22}$$

où a est le taux d'apprentissage (learning rate).

Remarque : La formule du gradient ressemble à celle de la régression linéaire !

Algorithme de la régression logistique

Input: Données $\mathbf{X}$, labels $\mathbf{y}$, taux d'apprentissage a, nombre d'itérations T

Output: Paramètres optimaux β

Initialiser β aléatoirement;

for $t = 1$ ***to*** T **do**

- $\mathbf{z} \leftarrow \mathbf{X}\beta$;
- $\hat{\mathbf{y}} \leftarrow \sigma(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}}$;
- $\nabla J \leftarrow \frac{1}{n}\mathbf{X}^T(\hat{\mathbf{y}} - \mathbf{y})$;
- $\beta \leftarrow \beta - a\nabla J$;
- Calculer et stocker $J(\beta)$ (optionnel);

end

return β;

Algorithm 1: Régression Logistique par Descente de Gradient

Seuil de décision

Problème : Comment convertir les probabilités en classes ?

Seuil par défaut : 0.5

- Si $P(y = 1|\mathbf{x}) \geq 0.5$: prédire classe 1
- Si $P(y = 1|\mathbf{x}) < 0.5$: prédire classe 0

Ajustement du seuil :

- Selon le contexte, on peut choisir un autre seuil
- **Seuil élevé (ex: 0.7)** : Moins de faux positifs, plus conservateur
- **Seuil faible (ex: 0.3)** : Moins de faux négatifs, plus permissif

Exemples :

- **Détection de fraude** : Seuil bas (préférer détecter toutes les fraudes, quitte à avoir des fausses alarmes)
- **Diagnostic médical grave** : Seuil bas (ne pas manquer un cas)
- **Filtrage spam** : Seuil modéré (équilibre)

Métriques d'évaluation : Matrice de confusion

Matrice de confusion :

	Prédit 0	**Prédit 1**
Réel 0	VN	FP
Réel 1	FN	VP

- **VP** : Vrais Positifs (True Positive)
- **VN** : Vrais Négatifs (True Negative)
- **FP** : Faux Positifs (False Positive) - Erreur Type I
- **FN** : Faux Négatifs (False Negative) - Erreur Type II

Exemple : Détection de spam

	Prédit Non-Spam	**Prédit Spam**
Non-Spam	85	5
Spam	10	100

- VP = 100 (spam détecté)
- VN = 85 (non-spam correct)
- FP = 5 (fausse alerte)
- FN = 10 (spam manqué)

Métriques d'évaluation : Formules

1. Accuracy (Exactitude) :

$$\text{Accuracy} = \frac{VP + VN}{VP + VN + FP + FN} \tag{23}$$

Proportion de prédictions correctes (tous types confondus).

2. Precision (Précision) :

$$\text{Precision} = \frac{VP}{VP + FP} \tag{24}$$

Parmi les prédictions positives, combien sont correctes ?

3. Recall (Rappel / Sensibilité) :

$$\text{Recall} = \frac{VP}{VP + FN} \tag{25}$$

Parmi les cas positifs réels, combien sont détectés ?

4. F1-Score :

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{} \tag{26}$$

Quand utiliser quelle métrique ?

- **Accuracy** :
  - Bon pour les classes équilibrées
  - Trompeur si déséquilibre (ex: 95% classe 0, 5% classe 1)
- **Precision** :
  - Important quand les **faux positifs** coûtent cher
  - Exemple : Diagnostic de maladie grave (ne pas alarmer inutilement)
- **Recall** :
  - Important quand les **faux négatifs** coûtent cher
  - Exemple : Détection de cancer (ne pas manquer un cas)
- **F1-Score** :
  - Équilibre entre precision et recall
  - Utile pour comparer des modèles

Compromis Precision-Recall : Améliorer l'un peut dégrader l'autre !

Courbe ROC (Receiver Operating Characteristic)

Définition :

- Graphique : Taux de Vrais Positifs (TPR) vs. Taux de Faux Positifs (FPR)
- TPR = Recall = $\frac{VP}{VP+FN}$
- FPR = $\frac{FP}{FP+VN}$
- Tracée pour différents seuils

AUC (Area Under Curve) :

- Aire sous la courbe ROC
- AUC = 1 : Classificateur parfait
- AUC = 0.5 : Hasard
- AUC > 0.8 : Bon modèle

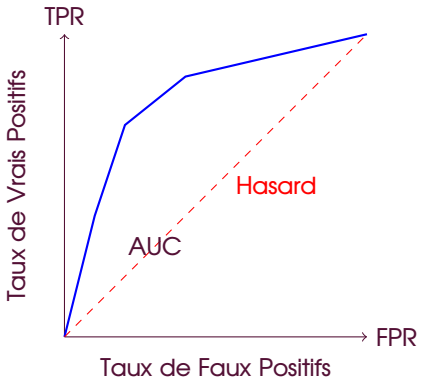


Figure: Exemple de courbe ROC

Régression logistique multiclasse

Stratégies pour plus de 2 classes :

1. One-vs-Rest (OvR) / One-vs-All :

- Entraîner K classificateurs binaires (un par classe)
- Classificateur k : classe k vs. toutes les autres
- Prédiction : choisir la classe avec la probabilité maximale

2. One-vs-One (OvO) :

- Entraîner $\frac{K(K-1)}{2}$ classificateurs
- Un classificateur pour chaque paire de classes
- Prédiction par vote majoritaire

3. Régression logistique multinomiale (Softmax) :

- Extension directe avec fonction softmax
- $P(y = k|\mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x}}}$

Avantages et inconvénients

Avantages :

- Simple et rapide à entraîner
- Interprétable (importance des features)
- Sortie probabiliste
- Fonctionne bien avec peu de données
- Peu de paramètres à ajuster
- Pas de problème de convergence locale (convexe)

Inconvénients :

- Suppose une relation linéaire (après transformation)
- Sensible aux features non pertinentes
- Nécessite feature engineering
- Mauvais pour relations complexes/non-linéaires
- Sensible aux valeurs aberrantes
- Moins performant que les modèles complexes

Quand l'utiliser ?

- Baseline pour tout problème de classification
- Relations linéaires ou quasi-linéaires
- Besoin d'interprétabilité

Exemple Python : Chargement des données

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             roc_curve, roc_auc_score)
from sklearn.datasets import load_breast_cancer

# Chargement du dataset Breast Cancer (classification binaire)
cancer = load_breast_cancer()
X = cancer.data
y = cancer.target  # 0 = maligne, 1 = benigne

print(f"Nombre d'observations : {X.shape[0]}")
print(f"Nombre de features : {X.shape[1]}")
print(f"Classes : {cancer.target_names}")
print(f"Distribution : {np.bincount(y)}")

# Separation des donnees
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
```

Exemple Python : Entraînement du modèle

```python
# Normalisation des donnees (important pour la regression logistique)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Creation et entra■nement du mod€le
model = LogisticRegression(max_iter=10000, random_state=42)
model.fit(X_train_scaled, y_train)

# Predictions
y_pred_train = model.predict(X_train_scaled)
y_pred_test = model.predict(X_test_scaled)

# Predictions probabilistes
y_pred_proba_test = model.predict_proba(X_test_scaled)[:, 1]

print(f"Intercept : {model.intercept_}")
print(f"Nombre de coefficients : {len(model.coef_[0])}")
print(f"Top 5 features par importance (valeur absolue) :")
feature_importance = pd.DataFrame({
    'feature': cancer.feature_names,
```

Exemple Python : Évaluation - Métriques

```python
# Calcul des metriques
train_acc = accuracy_score(y_train, y_pred_train)
test_acc = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test, y_pred_test)
recall = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)

print("=== PERFORMANCES ===")
print(f"\nAccuracy (entraînement) : {train_acc:.4f}")
print(f"Accuracy (test) : {test_acc:.4f}")
print(f"\nPrecision : {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1-Score : {f1:.4f}")

# Rapport de classification complet
print("\n=== RAPPORT DE CLASSIFICATION ===")
print(classification_report(y_test, y_pred_test,
                            target_names=cancer.target_names))

# Matrice de confusion
cm = confusion_matrix(y_test, y_pred_test)
print("\n=== MATRICE DE CONFUSION ===")
print(cm)
```

Exemple Python : Visualisation de la matrice de confusion

```python
import seaborn as sns

# Visualisation de la matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=cancer.target_names,
            yticklabels=cancer.target_names)
plt.xlabel('Prediction')
plt.ylabel('Valeur reelle')
plt.title('Matrice de confusion')
plt.show()

# Exemples de predictions
print("\n=== EXEMPLES DE PREDICTIONS ===")
for i in range(5):
    true_label = cancer.target_names[y_test.iloc[i] if isinstance(y_test, pd.Series)
                                     else y_test[i]]
    pred_label = cancer.target_names[y_pred_test[i]]
    proba = y_pred_proba_test[i]
    print(f"Observation {i+1}:")
    print(f"  Vrai : {true_label}")
    print(f"  Predit : {pred_label} (probabilite = {proba:.3f})")
```

Exemple Python : Courbe ROC

```python
# Calcul de la courbe ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_test)
auc = roc_auc_score(y_test, y_pred_proba_test)

# Visualisation
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, linewidth=2, label=f'ROC Curve (AUC = {auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--', label='Hasard (AUC = 0.5)')
plt.xlabel('Taux de Faux Positifs (FPR)')
plt.ylabel('Taux de Vrais Positifs (TPR)')
plt.title('Courbe ROC - Breast Cancer Classification')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

print(f"\nAUC Score : {auc:.4f}")

# Trouver le seuil optimal (Youden's index)
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print(f"Seuil optimal : {optimal_threshold:.3f}")
print(f"TPR au seuil optimal : {tpr[optimal_idx]:.3f}")
print(f"FPR au seuil optimal : {fpr[optimal_idx]:.3f}")
```

Exemple Python : Implémentation manuelle (bonus)

```python
# Implementation manuelle de la regression logistique
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def log_loss(y_true, y_pred):
    epsilon = 1e-15  # Pour eviter log(0)
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return -np.mean(y_true * np.log(y_pred) +
                    (1 - y_true) * np.log(1 - y_pred))

def gradient_descent_logistic(X, y, learning_rate=0.01, n_iterations=1000):
    n_samples, n_features = X.shape
    weights = np.zeros(n_features)
    bias = 0
    losses = []

    for i in range(n_iterations):
        # Predictions
        z = np.dot(X, weights) + bias
        y_pred = sigmoid(z)

        # Gradient
        dw = (1/n_samples) * np.dot(X.T, (y_pred - y))
```

Exemple Python : Comparaison implémentation manuelle

```python
# Entraînement avec notre implementation
weights_manual, bias_manual, losses = gradient_descent_logistic(
    X_train_scaled, y_train, learning_rate=0.1, n_iterations=1000
)

# Predictions
z_test = np.dot(X_test_scaled, weights_manual) + bias_manual
y_pred_proba_manual = sigmoid(z_test)
y_pred_manual = (y_pred_proba_manual >= 0.5).astype(int)

# Evaluation
acc_manual = accuracy_score(y_test, y_pred_manual)
print(f"Accuracy (implementation manuelle) : {acc_manual:.4f}")
print(f"Accuracy (sklearn) : {test_acc:.4f}")

# Visualisation de la convergence
plt.figure(figsize=(8, 5))
plt.plot(range(0, 1000, 100), losses, marker='o')
plt.xlabel('Iteration')
plt.ylabel('Log-Loss')
plt.title('Convergence de la descente de gradient')
plt.grid(True, alpha=0.3)
plt.show()
```

Exercice pratique

Dataset suggéré : Titanic, Iris (binaire), ou créez vos propres données

À faire :

1. Charger et explorer les données
2. Prétraiter les données (normalisation, encodage)
3. Diviser en train/test
4. Entraîner un modèle de régression logistique
5. Calculer toutes les métriques (accuracy, precision, recall, F1)
6. Afficher et interpréter la matrice de confusion
7. Tracer la courbe ROC et calculer l'AUC
8. Tester différents seuils de décision
9. Analyser les coefficients du modèle

Bonus :

- Implémenter la régression logistique from scratch
- Comparer avec d'autres modèles (KNN, Decision Tree)
- Faire du feature engineering

Résumé du chapitre

Concepts clés :

- Classification binaire
- Fonction sigmoïde
- Log-loss (entropie croisée)
- Descente de gradient
- Seuil de décision
- Matrice de confusion
- Precision, Recall, F1
- Courbe ROC et AUC

Points importants :

- Normaliser les features
- Choisir le bon seuil selon le contexte
- Ne pas se fier uniquement à l'accuracy

Formules essentielles :

- $\sigma(z) = \frac{1}{1+e^{-z}}$
- $\hat{y} = \sigma(\beta^T \mathbf{x})$
- $J = -\frac{1}{n} \sum [y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$
- Precision $= \frac{VP}{VP+FP}$
- Recall $= \frac{VP}{VP+FN}$

Pour aller plus loin

Extensions et méthodes avancées :

- **Régularisation** : L1 (Lasso), L2 (Ridge) pour éviter le surapprentissage
- **Régression logistique multinomiale** : Plus de 2 classes (softmax)
- **Features polynomiaux** : Capturer des relations non-linéaires
- **Interactions entre features** : Produits de variables
- **Calibration des probabilités** : Améliorer les estimations probabilistes
- **Déséquilibre de classes** : SMOTE, sous-échantillonnage, poids des classes

Autres algorithmes de classification :

- K-Nearest Neighbors (KNN)
- Support Vector Machines (SVM)
- Arbres de décision et Random Forest
- Gradient Boosting (XGBoost, LightGBM)
- Réseaux de neurones

Apprentissage non supervisé

Apprentissage non supervisé : Clustering

Objectifs du chapitre :

1. Comprendre l'apprentissage non supervisé
2. Maîtriser l'algorithme K-Means
3. Découvrir le clustering hiérarchique
4. Introduction à DBSCAN
5. Évaluer la qualité du clustering
6. Réduction de dimensionnalité (PCA)

Supervisé vs. Non supervisé

Apprentissage supervisé :

- Données étiquetées : (X, y)
- On connaît la "bonne réponse"
- Objectif : prédire y pour de nouvelles données
- Exemples : régression, classification

Applications :

- Prédire le prix d'une maison
- Détecter les emails spam
- Diagnostiquer une maladie

Apprentissage non supervisé :

- Données non étiquetées : seulement X
- Pas de "bonne réponse" connue
- Objectif : découvrir des structures cachées
- Exemples : clustering, réduction de dimension

Applications :

- Segmentation de clients
- Détection d'anomalies
- Compression de données

Qu'est-ce que le clustering ?

Définition

Le clustering (ou partitionnement) est une technique qui consiste à regrouper des données similaires en groupes appelés "clusters", sans connaissance préalable des catégories.

Principe :

- Les objets d'un même cluster sont **similaires** entre eux
- Les objets de clusters différents sont **dissimilaires**
- Aucune étiquette fournie au départ

Applications pratiques :

- **Marketing** : Segmentation de clients selon leurs comportements d'achat
- **Biologie** : Regroupement de gènes ayant des expressions similaires
- **Médecine** : Identification de sous-types de maladies
- **Image** : Compression d'images, segmentation
- **Détection d'anomalies** : Points isolés = anomalies potentielles

Types de méthodes de clustering

1. **Méthodes par partitionnement :**
   - **K-Means** : Partitionne en K clusters avec des centroïdes
   - **K-Medoids (PAM)** : Utilise des points réels comme centres
2. **Méthodes hiérarchiques :**
   - **Agglomératif** : Fusion progressive des clusters
   - **Divisif** : Division progressive des clusters
3. **Méthodes basées sur la densité :**
   - **DBSCAN** : Détecte les zones denses
   - Peut identifier des formes arbitraires et des outliers
4. **Autres :**
   - Clustering flou (Fuzzy C-Means)
   - Modèles de mélange gaussien (GMM)

K-Means : Principe

Objectif

Partitionner n observations en K clusters de manière à minimiser la variance intra-cluster.

Notations :

- $X = \{x_1, x_2, \ldots, x_n\}$: ensemble des points
- K : nombre de clusters (fixé à l'avance)
- $C = \{C_1, C_2, \ldots, C_K\}$: les K clusters
- μ_k : centroïde du cluster C_k

Fonction objectif (inertie) :

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \tag{27}$$

But : Minimiser J (somme des distances au carré entre chaque point et son centroïde)

Algorithme K-Means

Input: Données X, nombre de clusters K, nombre d'itérations max T

Output: Clusters C et centroïdes μ

Initialisation : Choisir K centroïdes aléatoires $\mu_1, \ldots, \mu_K$;

for $t = 1$ ***to*** T **do**

  Étape 1 - Affectation :;

  for *chaque point* x_i **do**

    Assigner x_i au cluster du centroïde le plus proche :;

    $C_k = \{x_i : \|x_i - \mu_k\| \leq \|x_i - \mu_j\|, \forall j\}$;

  end

  Étape 2 - Mise à jour :;

  for *chaque cluster* C_k **do**

    Recalculer le centroïde :;

    $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$;

  end

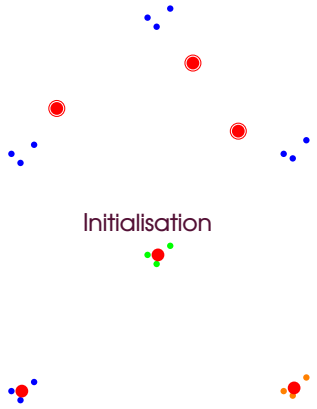
  if *convergence (centroïdes ne changent plus)* **then**

LISV Laboratoire d'ingénierie des systèmes de Versailles — UVSQ université PARIS-SACLAY

K-Means : Illustration

Étapes de l'algorithme :

1. **Initialisation :**
   - Placer K centroïdes aléatoirement
2. **Affectation :**
   - Chaque point rejoint le cluster du centroïde le plus proche
3. **Mise à jour :**
   - Recalculer les centroïdes (moyenne des points du cluster)
4. **Répéter :**
   - Jusqu'à convergence



LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

K-Means : Choix de K (méthode du coude)

Problème : Comment choisir le bon nombre de clusters K ?

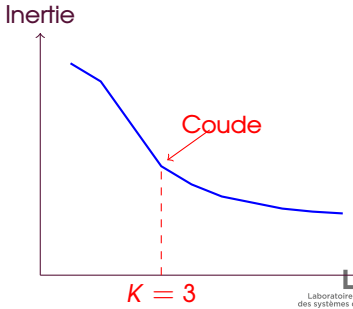
Méthode du coude (Elbow method) :

1. Calculer l'inertie pour différentes valeurs de K (1, 2, 3, ...)
2. Tracer l'inertie en fonction de K
3. Chercher le "coude" : point où l'inertie décroît moins rapidement

Inertie :

$$I = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - \mu_k||^2 \qquad (28)$$

- Plus K augmente, plus l'inertie diminue
- $K = n$: inertie = 0 (chaque point = cluster)
- Chercher le compromis optimal



K-Means : Score de silhouette

Mesure la qualité du clustering :

Pour un point i :

- $a(i)$: distance moyenne aux autres points du même cluster
- $b(i)$: distance moyenne aux points du cluster le plus proche

Coefficient de silhouette :

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{29}$$

Interprétation :

- $s(i) \approx 1$: Point bien affecté à son cluster
- $s(i) \approx 0$: Point à la frontière entre deux clusters
- $s(i) < 0$: Point probablement mal affecté

Score de silhouette global :

$$S = \frac{1}{n} \sum_{i=1}^{n} s(i) \tag{30}$$

Choisir le K qui maximise S (généralement entre -1 et 1).

K-Means : Avantages et limites

Avantages :

- Simple et rapide
- Efficace sur grands ensembles
- Complexité : $O(n \cdot K \cdot d \cdot T)$
- Facile à implémenter
- Fonctionne bien si clusters sphériques

Limites :

- Nécessite de fixer K à l'avance
- Sensible à l'initialisation
- Suppose clusters sphériques
- Sensible aux outliers
- Mauvais pour formes non-convexes
- Convergence vers minimum local

Solutions :

- **K-Means++** : Meilleure initialisation des centroïdes
- **Mini-Batch K-Means** : Plus rapide pour grands datasets
- **Exécutions multiples** : Lancer plusieurs fois avec initialisations différentes

Clustering hiérarchique

Principe

Construire une hiérarchie de clusters sous forme d'arbre (dendrogramme), sans fixer K à l'avance.

Deux approches :

1. Agglomératif (bottom-up) :

- Départ : chaque point = un cluster
- Fusion itérative des clusters les plus proches
- Jusqu'à n'avoir qu'un seul cluster

2. Divisif (top-down) :

- Départ : tous les points dans un cluster
- Division itérative en sous-clusters
- Moins utilisé (plus coûteux)

Résultat : Un dendrogramme permettant de choisir K après coup

Clustering hiérarchique : Algorithme agglomératif

Input: Données X, critère de liaison
Output: Dendrogramme
Initialiser : Chaque point x_i forme un cluster C_i;
while *Il reste plus d'un cluster* **do**
    Trouver les deux clusters C_i et C_j les plus proches;
    Fusionner C_i et C_j en un nouveau cluster C_{new};
    Enregistrer la fusion dans le dendrogramme;
    Mettre à jour la matrice de distances;
end
return Dendrogramme;

Algorithm 3: Clustering Hiérarchique Agglomératif

Question clé : Comment mesurer la distance entre deux clusters ?

Critères de liaison (linkage)

Distance entre clusters C_i et C_j :

1. Simple (Single linkage) :

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \tag{31}$$

Distance entre les points les plus proches. Sensible aux outliers ("chaining").

2. Complete (Complete linkage) :

$$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y) \tag{32}$$

Distance entre les points les plus éloignés. Forme des clusters compacts.

3. Average (Average linkage) :

$$d(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y) \tag{33}$$

Distance moyenne entre tous les points. Compromis équilibré.

4. Ward : Minimise la variance intra-cluster. Le plus utilisé en pratique.

Dendrogramme

Lecture du dendrogramme :

- Axe vertical : distance de fusion
- Couper à une hauteur = choisir K
- Plus la fusion est haute, plus les clusters sont différents

Avantages :

- Pas besoin de fixer K au départ
- Visualisation de la hiérarchie
- Déterministe (pas d'aléatoire)

Limites :

- Complexité : $O(n^3)$ ou $O(n^2 \log n)$
- Pas adapté aux grands datasets
- Choix du critère de liaison important

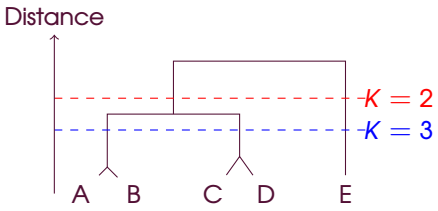


Figure: Exemple de dendrogramme

DBSCAN (Density-Based Spatial Clustering)

Principe

Identifier les régions denses et regrouper les points dans ces régions. Les points isolés sont considérés comme du bruit (outliers).

Paramètres :

- ε (eps) : Rayon de voisinage
- MinPts : Nombre minimum de points pour former une région dense

Types de points :

- **Core point** : Au moins MinPts voisins dans un rayon ε
- **Border point** : Moins de MinPts voisins, mais dans le voisinage d'un core point
- **Noise point** : Ni core ni border (outlier)

Construction des clusters :

- Connecter les core points voisins
- Ajouter les border points aux clusters
- Isoler les noise points

DBSCAN : principe algorithmique

Entrées : données X, rayon ε, MinPts

1. Marquer tous les points comme non visités
2. Pour chaque point non visité p :
   - Trouver les voisins de p à distance ε
   - Si $|N(p)| < \text{MinPts}$: p est du bruit
   - Sinon : créer un nouveau cluster
   - Étendre le cluster via les voisins denses
3. Répéter jusqu'à ce que tous les points soient visités

Idée clé

Les clusters sont définis par la **densité**, pas par une forme géométrique.

DBSCAN : Avantages et limites

Avantages :

- Détecte des formes arbitraires
- Pas besoin de fixer K
- Identifie les outliers
- Robuste au bruit
- Peut trouver des clusters de densités variables

Limites :

- Sensible aux paramètres ε et MinPts
- Mauvais pour densités très variables
- Complexité : $O(n \log n)$ avec index spatial
- Difficulté en haute dimension

Comparaison K-Means vs DBSCAN :

- **K-Means** : Rapide, clusters sphériques, nécessite K
- **DBSCAN** : Formes arbitraires, détecte outliers, sensible aux paramètres

Quand utiliser DBSCAN ?

- Clusters de formes complexes
- Présence d'outliers importante
- Densité variable

Réduction de dimensionnalité : PCA

Analyse en Composantes Principales (PCA)

Technique pour réduire le nombre de dimensions tout en préservant le maximum de variance.

Objectif :

- Transformer p variables en k composantes principales ($k < p$)
- Les composantes sont des combinaisons linéaires des variables originales
- Préserver le maximum d'information

Applications :

- Visualisation de données en 2D ou 3D
- Réduction de bruit
- Accélération des algorithmes
- Compression de données
- Preprocessing avant clustering

Principe : Trouver les directions de variance maximale dans les données.

PCA : Algorithme

Étapes de la PCA :

1. **Centrer les données :**

$$X_{centree} = X - \bar{X} \tag{34}$$

2. **Calculer la matrice de covariance :**

$$\Sigma = \frac{1}{n-1} X_{centree}^T X_{centree} \tag{35}$$

3. **Calculer les valeurs propres et vecteurs propres :**
   - Décomposition en valeurs propres de Σ
   - Les vecteurs propres sont les composantes principales
   - Les valeurs propres indiquent la variance expliquée
4. **Trier par variance décroissante :**
   - Ordonner les vecteurs propres par valeurs propres décroissantes
5. **Projeter les données :**

$$X_{reduite} = X_{centee} \cdot W_k \tag{36}$$

où W_k contient les k premiers vecteurs propres

PCA : Variance expliquée

Choisir le nombre de composantes :

Variance expliquée par la composante i :

$$\text{Variance}_i = \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j} \tag{37}$$

Variance cumulée :

$$\text{Variance cumulée}_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j} \tag{38}$$

Critère de choix :

- Garder k composantes qui expliquent 80-95% de la variance
- Méthode du coude sur le graphique de variance

Exemple :

- PC1 : 45% de variance, PC2 : 25% de variance
- PC3 : 15% de variance
- $\rightarrow$ 2 composantes = 70%, 3 composantes = 85%

Exemple Python : K-Means (choix de K)

```
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)

inertias = []
sil = []

for k in range(2, 11):
    km = KMeans(n_clusters=k, random_state=42)
    labels = km.fit_predict(X)
    inertias.append(km.inertia_)
    sil.append(silhouette_score(X, labels))

print(inertias)
print(sil)
```

Point clé

Le coude minimise l'inertie, la silhouette mesure la qualité des clusters.

Exemple Python : K-Means (clustering final)

```
K = 4
kmeans = KMeans(n_clusters=K, random_state=42)
labels = kmeans.fit_predict(X)

print("Inertie :", kmeans.inertia_)
print("Tailles des clusters :", np.bincount(labels))
```

Point clé

Les centroïdes résument chaque cluster et dépendent fortement du choix de K.

Exemple Python : Clustering hiérarchique

```python
from scipy.cluster.hierarchy import linkage
from sklearn.cluster import AgglomerativeClustering

# Lien hierarchique (Ward)
Z = linkage(X, method='ward')

# Clustering agglomeratif
agg = AgglomerativeClustering(
    n_clusters=4, linkage='ward'
)

labels = agg.fit_predict(X)

print("Taille des clusters :", np.bincount(labels))
```

Point clé

Le dendrogramme sert à choisir le nombre de clusters, le modèle final est appliqué avec `AgglomerativeClustering`.

Exemple Python : DBSCAN

```python
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)

n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = (labels == -1).sum()

print("Clusters :", n_clusters)
print("Bruit :", n_noise)
```

Point clé

DBSCAN détecte automatiquement le nombre de clusters et les points de bruit.

Exemple Python : PCA

```
# Standardisation (indispensable pour PCA)
X_scaled = StandardScaler().fit_transform(X)

# PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Variance expliquee
var = pca.explained_variance_ratio_
cum_var = var.cumsum()

print(var)
print(cum_var)
```

Point clé

On choisit le nombre de composantes pour expliquer une large part de la variance (ex. 90{95%).

Exemple Python : PCA + K-Means

```python
# PCA a 2 dimensions
X_pca = PCA(n_components=2).fit_transform(X_scaled)

# K-Means sur donnees reduites
kmeans = KMeans(n_clusters=4, random_state=42)
labels = kmeans.fit_predict(X_pca)

print("Variance expliquee :",
      PCA(n_components=2).fit(X_scaled)
         .explained_variance_ratio_.sum())
```

Point clé

La PCA facilite la visualisation et peut améliorer le clustering, mais elle peut aussi faire perdre de l'information.

Exercice pratique

Dataset suggéré : Iris, Mall Customers, ou données synthétiques

Préparation

- Charger et explorer les données
- Normaliser les features

K-Means

- Choisir K (coude, silhouette)
- Visualiser les clusters
- Interpréter les centroïdes

Clustering hiérarchique

- Dendrogramme
- Comparaison avec K-Means

DBSCAN

- Tester plusieurs paramètres
- Identifier les outliers

PCA

- Réduction en 2D
- Visualisation des clusters

Bonus

- Comparer K-Means / Hiérarchique / DBSCAN
- K-Means from scratch

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Résumé du chapitre

Concepts clés :

- Apprentissage non supervisé
- Clustering
- K-Means (centroïdes)
- Hiérarchique (dendrogramme)
- DBSCAN (densité)
- Score de silhouette
- PCA (réduction dimension)

Formules essentielles :

- Inertie : $\sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$
- Silhouette : $s(i) = \frac{b(i)-a(i)}{\max(a(i),b(i))}$
- Centroïde : $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$

Comparaison des méthodes :

- **K-Means** : Rapide, nécessite K, clusters sphériques
- **Hiérarchique** : Pas besoin de K, coûteux, visualisation
- **DBSCAN** : Formes arbitraires, détecte outliers, sensible aux paramètres

Points importants :

- Toujours normaliser les données

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Pour aller plus loin

Autres méthodes de clustering :

- **Gaussian Mixture Models (GMM)** : Clustering probabiliste
- **Spectral Clustering** : Basé sur la théorie des graphes
- **Mean Shift** : Basé sur la densité, pas besoin de K
- **OPTICS** : Extension de DBSCAN pour densités variables
- **Fuzzy C-Means** : Appartenance floue aux clusters

Autres techniques de réduction de dimension :

- **t-SNE** : Visualisation non-linéaire en 2D/3D
- **UMAP** : Alternative moderne à t-SNE
- **Autoencoders** : Réduction par réseaux de neurones
- **LDA** : Linear Discriminant Analysis (supervisé)

Applications avancées :

- Détection d'anomalies (Isolation Forest, One-Class SVM)
- Systèmes de recommandation
- Segmentation d'images

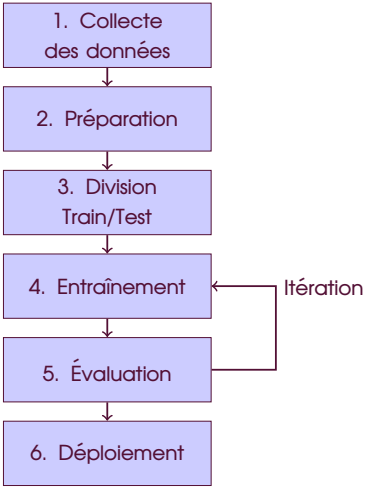
Les outils du Machine Learning

Les outils et bonnes pratiques du Machine Learning

Objectifs du chapitre :

1. Maîtriser la préparation des données
2. Comprendre la validation croisée
3. Analyser le compromis biais-variance
4. Optimiser les hyperparamètres
5. Créer des pipelines reproductibles
6. Appliquer les bonnes pratiques du ML

Le workflow complet du Machine Learning



Ce chapitre couvre les étapes 2 à 5 en détail

Préparation des données : Vue d'ensemble

Importance

"Garbage in, garbage out" - La qualité des données détermine la qualité du modèle.

Étapes principales :

1. **Nettoyage** : Supprimer les doublons, corriger les erreurs
2. **Valeurs manquantes** : Imputation ou suppression
3. **Outliers** : Détection et traitement
4. **Normalisation** : Mise à l'échelle des features
5. **Encodage** : Transformation des variables catégorielles
6. **Feature engineering** : Création de nouvelles variables

Règle d'or :

- 80% du temps = préparation des données
- 20% du temps = modélisation

Gestion des valeurs manquantes

Causes des valeurs manquantes :

- Erreurs de saisie
- Données non collectées
- Fusion de datasets
- Capteurs défaillants

Stratégies de traitement :

1. Suppression :

- Supprimer les lignes (si peu nombreuses)
- Supprimer les colonnes (si trop de manquants)

2. Imputation simple :

- Moyenne / Médiane (numériques)
- Mode (catégorielles)
- Valeur constante (0, "Unknown")

3. Imputation avancée :

- KNN Imputer
- Régression
- Interpolation (séries temporelles)

Normalisation et standardisation

Pourquoi normaliser ?

- Algorithmes sensibles aux échelles (KNN, SVM, régression avec régularisation)
- Convergence plus rapide (descente de gradient)
- Comparaison des coefficients

Méthodes principales :

1. Standardisation (Z-score) :

$$x_{scaled} = \frac{x - \mu}{\sigma} \tag{39}$$

- Moyenne = 0, écart-type = 1
- Préserve les outliers
- Utilisé par : SVM, régression logistique, PCA

2. Normalisation Min-Max :

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{40}$$

Encodage des variables catégorielles

Problème : Les algorithmes ML travaillent avec des nombres, pas du texte.

1. **Label Encoding :**
   - Transformer en entiers : ("rouge", "vert", "bleu") → (0, 1, 2)
   - **Attention :** Crée un ordre artificiel !
   - Utiliser pour variables ordinales (petit < moyen < grand)
2. **One-Hot Encoding :**
   - Créer une colonne binaire par catégorie
   - Exemple : "couleur" → ("couleur_rouge", "couleur_vert", "couleur_bleu")
   - Pas d'ordre artificiel
   - Inconvénient : augmente la dimensionnalité
3. **Target Encoding :**
   - Remplacer par la moyenne de la cible pour cette catégorie
   - Risque de data leakage
   - Nécessite validation croisée

Détection et traitement des outliers

Méthodes de détection :

1. Méthode statistique :

- Z-score : $|z| > 3$
- IQR : en dehors de $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$

2. Visualisation :

- Boxplots
- Scatter plots
- Histogrammes

3. Algorithmes ML :

- Isolation Forest
- Local Outlier Factor (LOF)
- DBSCAN

Stratégies de traitement : 1. Suppression :

- Si erreur de mesure
- Si très peu nombreux

2. Transformation :

- Log transformation
- Winsorization (cap values)
- Robust scaling

3. Conservation :

- Si information pertinente
- Utiliser algorithmes robustes

Question clé : L'outlier est-il une erreur ou une information ?

Division des données : Train/Test/Validation

Ensembles de données :

1. Entraînement (Train) :

- 60-80% des données
- Apprendre les paramètres

2. Validation :

- 10-20% des données
- Ajuster les hyperparamètres
- Sélection de modèle

3. Test :

- 10-20% des données
- Évaluation finale
- Ne jamais utiliser pour l'entraînement !

Données complètes

Règles importantes :

- Mélanger les données avant division
- Stratification si classes déséquilibrées
- Test set = donnée "jamais vue"
- Ne pas normaliser avant division !

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

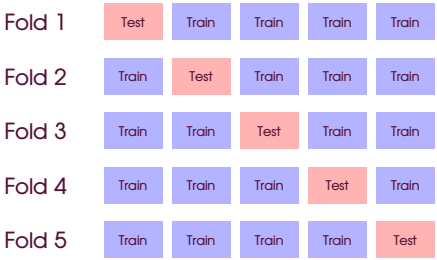
Validation croisée (Cross-Validation)

Objectif

Obtenir une estimation plus robuste des performances en utilisant plusieurs divisions train/test différentes.

K-Fold Cross-Validation :

1. Diviser les données en K plis (folds)
2. Pour chaque pli k :
   - Utiliser le pli k comme test
   - Utiliser les $K - 1$ autres plis comme train
   - Entraîner et évaluer le modèle
3. Calculer la moyenne des K scores



5-Fold Cross-Validation

Types de validation croisée

1. K-Fold CV

- Standard : $K = 5$ ou $K = 10$
- Bon compromis biais/variance

2. Stratified K-Fold

- Proportion des classes préservée
- Essentiel si classes déséquilibrées
- Classification

3. Leave-One-Out (LOOCV)

- $K = n$ (1 échantillon test)
- Faible biais, forte variance
- Coût de calcul élevé

4. Time Series CV

- Respect de l'ordre temporel
- Train = passé, test = futur
- Pas de mélange aléatoire

Pourquoi utiliser la CV ?

- Meilleure utilisation des données
- Estimation plus robuste
- Détection du surapprentissage

Compromis biais--variance

Définitions

Biais : hypothèses trop simplificatrices **Variance** : sensibilité aux données d'entraînement

Biais élevé (underfitting)

- Modèle trop simple
- Patterns non capturés
- Performances faibles train/test
- Ex. : modèle linéaire inadapté

Solutions

- Modèle plus complexe
- Plus de features
- Moins de régularisation

Variance élevée (overfitting)

- Modèle trop complexe
- Apprentissage du bruit
- Bon train, mauvais test
- Ex. : polynôme trop élevé

Solutions

- Plus de données
- Régularisation (L1, L2)
- Simplifier le modèle
- Early stopping

Diagnostic : Courbes d'apprentissage

Courbe d'apprentissage

- Erreur vs taille du dataset
- Courbes : train et validation

Interprétation : Biais élevé

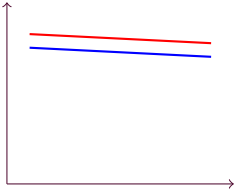
- Erreurs élevées, Convergent vers une valeur haute
- Plus de données inutile

Variance élevée

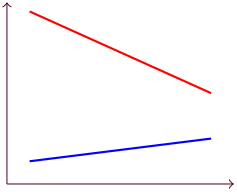
- Grand écart train/validation
- Plus de données aide

Bon ajustement

- Erreurs faibles , Écart faible



Biais élevé



Variance élevée

Train Validation

Hyperparamètres vs. Paramètres

Paramètres :

- Appris pendant l'entraînement
- Optimisés automatiquement
- Exemples :
  - Coefficients régression : β
  - Poids réseau neurones : w
  - Centroïdes K-Means : μ

Impact des hyperparamètres :

- Affectent directement les performances
- Déterminent le compromis biais-variance
- Nécessitent tuning systématique

Hyperparamètres :

- Fixés avant l'entraînement
- Contrôlent l'apprentissage
- Nécessitent optimisation manuelle
- Exemples :
  - Learning rate : a
  - Nombre de clusters : K
  - Profondeur arbre : max_depth
  - Régularisation : λ
  - Nombre de voisins : k (KNN)

Optimisation des hyperparamètres : Grid Search

Principe

Tester toutes les combinaisons possibles d'hyperparamètres dans une grille prédéfinie.

Algorithme :

1. Définir une grille d'hyperparamètres
2. Pour chaque combinaison :
   - Entraîner le modèle avec CV
   - Calculer le score moyen
3. Sélectionner la meilleure combinaison
4. Réentraîner sur toutes les données

Exemple :

- Régression logistique : $C \in \{0.001, 0.01, 0.1, 1, 10, 100\}$
- K-Means : $K \in \{2, 3, 4, 5, 6, 7, 8\}$
- SVM : $C \in \{0.1, 1, 10\}$, kernel $\in$ {linear, rbf}

Avantages : Exhaustif, trouve l'optimum global

Inconvénients : Très coûteux (complexité exponentielle)

Optimisation : Random Search

Principe

Échantillonner aléatoirement des combinaisons d'hyperparamètres sur un budget fixé.

Avantages vs Grid Search

- Plus rapide
- Meilleure exploration
- Adapté aux espaces continus
- Facilement parallélisable

Quand utiliser ?

- **Grid** : peu de paramètres, discret
- **Random** : grand espace, nombreux paramètres
- **Bayesian** : budget limité

Méthodes avancées

- Bayesian Optimization (Optuna, Hyperopt)
- Algorithmes génétiques
- AutoML (Auto-sklearn, TPOT)

Feature Engineering

Définition

Création, transformation et sélection de features pour améliorer les performances du modèle.

1. Création de features

- Combinaisons : produit, ratio, différence
- Agrégations : somme, moyenne, max, min
- Transformations : log, sqrt, polynômes
- Variables temporelles : jour, mois, jour semaine
- Binning : discrétisation

2. Sélection de features

- **Filter** : corrélation, chi-square, ANOVA
- **Wrapper** : forward/backward, RFE
- **Embedded** : L1 (Lasso), arbres

Pipelines sklearn

Objectif

Enchaîner plusieurs étapes de preprocessing et modélisation dans un objet unique, évitant les erreurs et garantissant la reproductibilité.

Avantages des Pipelines :

- **Évite les fuites de données** : Le preprocessing est fait correctement sur train/test
- **Code plus propre** : Toutes les étapes en un seul objet
- **Grid Search facilité** : Optimiser preprocessing et modèle ensemble
- **Reproductibilité** : Mêmes transformations garanties
- **Déploiement simplifié** : Un seul objet à sauvegarder

Structure typique :

1. Imputation des valeurs manquantes
2. Encodage des variables catégorielles
3. Normalisation/Standardisation
4. Modèle ML

Bonnes pratiques du Machine Learning

1. Données

- Diviser train/test **avant** preprocessing
- Ne jamais toucher au test set avant l'évaluation finale
- Validation croisée pour le tuning
- Stratification si classes déséquilibrées

2. Preprocessing

- Fit sur train uniquement
- Transform sur train et test
- Sauvegarder les transformations
- Documenter les étapes

3. Modélisation

- Commencer par une baseline simple
- Augmenter la complexité progressivement
- Comparer plusieurs modèles
- Choisir des métriques adaptées

4. Évaluation

- Validation croisée systématique
- Analyse des erreurs
- Vérification des hypothèses

Sauvegarde et chargement de modèles

Pourquoi sauvegarder ?

- Éviter le réentraînement
- Déploiement en production
- Partage entre utilisateurs
- Versionnage des modèles

Méthodes en Python

1. Pickle

- Module Python natif
- Sérialisation générique
- Sensible aux versions

2. Joblib (recommandé)

- Optimisé pour NumPy / sklearn
- Plus efficace pour objets volumineux
- Compression intégrée

3. Formats spécifiques

- ONNX (interopérabilité)
- TensorFlow SavedModel
- PyTorch (.pt)

Exemple Python : Validation croisée

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Definition des modèles
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
}

# Validation croisee stratifiee
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Evaluation de chaque modèle
for name, model in models.items():
    scores = cross_val_score(model, X_train_processed, y_train,
                             cv=cv, scoring='accuracy')
    print(f"{name}:")
    print(f"  Moyenne: {scores.mean():.4f}")
    print(f"  Ecart-type: {scores.std():.4f}")
```

LISV Laboratoire d'Ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Exemple Python : Grid Search

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Definition de la grille d'hyperparamÃ¨tres
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10]
}
# Grid Search avec validation croisee
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=5, scoring='accuracy', n_jobs=-1
)
# Entrainement
grid_search.fit(X_train_processed, y_train)
# Resultats
print(f"Meilleurs paramÃ¨tres: {grid_search.best_params_}")
print(f"Meilleur score CV: {grid_search.best_score_:.4f}")
print(f"Score test: {grid_search.score(X_test_processed, y_test):.4f}")
```

Exemple Python : Pipeline complet

```python
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier

# Transformations pour variables numeriques
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Transformations pour variables categorielles
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# Combiner les transformations
preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])
```

Exemple Python : Grid Search avec Pipeline

```python
from sklearn.model_selection import GridSearchCV

# Grille d'hyperparamètres pour le pipeline
# Notation: 'nom_etape__nom_parametre'
param_grid_pipeline = {
    'preprocessor__num__imputer__strategy': ['mean', 'median'],
    'classifier__n_estimators': [50, 100, 200],
    'classifier__max_depth': [10, 15, None],
    'classifier__min_samples_split': [2, 5]
}
# Grid Search sur le pipeline complet
grid = GridSearchCV(
    pipeline, param_grid_pipeline,
    cv=5, scoring='accuracy', n_jobs=-1
)
# Entrainement
grid.fit(X_train, y_train)
# Resultats
print(f"Meilleurs paramètres: {grid.best_params_}")
print(f"Meilleur score CV: {grid.best_score_:.4f}")
print(f"Score test: {grid.score(X_test, y_test):.4f}")

# Le pipeline garantit: pas de data leakage!
```

Exemple Python : Courbes d'apprentissage

```python
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np
# Calcul des courbes d'apprentissage
train_sizes, train_scores, val_scores = learning_curve(
    pipeline, X_train, y_train, cv=5, train_sizes=np.linspace(0.1, 1.0, 10),
    scoring='accuracy', n_jobs=-1)
# Moyennes et ecarts-types
train_mean = train_scores.mean(axis=1)
train_std = train_scores.std(axis=1)
val_mean = val_scores.mean(axis=1)
val_std = val_scores.std(axis=1)

# Visualisation
plt.plot(train_sizes, train_mean, 'o-', color='blue', label='Train')
plt.plot(train_sizes, val_mean, 'o-', color='red', label='Validation')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, colo
plt.fill_between(train_sizes, val_mean-val_std, val_mean+val_std, alpha=0.1, color='red')
plt.xlabel('Taille dataset entrainement')
plt.ylabel('Accuracy')
plt.grid(True, alpha=0.3)
plt.show()
```

Exemple Python : Sauvegarde et chargement

```python
import joblib
import pickle
from datetime import datetime

# ===== SAUVEGARDE =====

# Methode 1: joblib (recommande pour sklearn)
joblib.dump(grid.best_estimator_, 'model.joblib')
print("Modèle sauvegarde: model.joblib")

# Methode 2: pickle (standard Python)
with open('model.pkl', 'wb') as f:
    pickle.dump(grid.best_estimator_, f)

# Sauvegarder aussi les metadonnees
metadata = {
    'model_type': 'RandomForestClassifier',
    'best_params': grid.best_params_,
    'cv_score': grid.best_score_,
    'test_score': grid.score(X_test, y_test),
    'date': datetime.now().isoformat()
}
joblib.dump(metadata, 'model_metadata.joblib')
```

Exemple Python : Sauvegarde et chargement

```python
import joblib
import pickle
from datetime import datetime


# ===== CHARGEMENT =====

# Charger le modèle
loaded_model = joblib.load('model.joblib')
loaded_metadata = joblib.load('model_metadata.joblib')

# Utiliser le modèle chargé
predictions = loaded_model.predict(X_test)
score = loaded_model.score(X_test, y_test)
print(f"Score du modèle chargé: {score:.4f}")
```

Exercice pratique complet

Dataset suggéré : Titanic, Adult Income, ou Bank Marketing

Objectif : Construire un pipeline ML complet de A à Z

À faire :

1. **Exploration** : Analyser les données, valeurs manquantes, types
2. **Preprocessing** :
   - Gérer les valeurs manquantes
   - Détecter et traiter les outliers
   - Encoder les variables catégorielles
   - Normaliser les variables numériques
3. **Modélisation** :
   - Créer un pipeline complet
   - Tester plusieurs modèles avec CV
   - Optimiser avec Grid Search
4. **Évaluation** :
   - Tracer les courbes d'apprentissage
   - Analyser biais-variance
   - Évaluer sur test set