**Annexe : Machine Learning (Apprentissage Automatique)**

**Objectifs :**

* Comprendre le Machine Learning.
* Implémenter des algorithmes supervisés et non-supervisés.
* Évaluer la performance des modèles.

1. **Introduction au Machine Learning**

Le **Machine Learning** (ou **apprentissage automatique**) est un sous-ensemble de l'intelligence artificielle (IA) qui permet aux ordinateurs d'**apprendre à partir de données** et d'**améliorer leurs performances** sans être explicitement programmés. L'idée principale est de donner aux systèmes la capacité d'identifier des **modèles** dans les données et de faire des prédictions ou des décisions en se basant sur ces modèles.

**1. Types de Machine Learning**

Il existe trois principaux types d'apprentissage automatique :

**a) Apprentissage Supervisé (Supervised Learning)**

* **Principe :** L'algorithme apprend à partir de données étiquetées (avec une sortie connue). Il construit un modèle à partir des **entrées** et des **sorties** connues et utilise ce modèle pour prédire des sorties sur de nouvelles données.
* **Exemples :**
  + **Classification :** Attribuer une étiquette à un élément (ex. : spam ou non-spam pour les emails).
  + **Régression :** Prédire une valeur continue (ex. : prédire le prix d'une maison à partir de ses caractéristiques).
* **Exemple d'algorithmes :** Régression linéaire, régression logistique, forêts aléatoires (Random Forest), machines à vecteurs de support (SVM), réseaux neuronaux.

**b) Apprentissage Non Supervisé (Unsupervised Learning)**

* **Principe :** L'algorithme apprend à partir de données non étiquetées. Il cherche des **structures cachées** ou des **patterns** dans les données sans avoir de sortie spécifiée.
* **Exemples :**
  + **Clustering (regroupement) :** Regrouper des éléments similaires dans des clusters (ex. : segmentation de clients en marketing).
  + **Réduction de dimensionnalité :** Réduire le nombre de variables tout en préservant l'information (ex. : PCA).
* **Exemple d'algorithmes :** K-means, analyse en composantes principales (PCA), algorithmes de clustering hiérarchique.

**c) Apprentissage par Renforcement (Reinforcement Learning)**

* **Principe :** L'algorithme apprend en **interagissant avec son environnement**. Il prend des actions et reçoit des **récompenses** ou des **sanctions**. L'objectif est de maximiser la récompense cumulée à long terme.
* **Exemples :**
  + **Jeux vidéo :** Un agent apprend à jouer à un jeu (ex. : AlphaGo de DeepMind).
  + **Robotique :** Un robot apprend à accomplir des tâches en s'améliorant au fur et à mesure.
* **Exemple d'algorithmes :** Q-learning, Deep Q-Network (DQN), Algorithmes de politique.

**2. Étapes du Processus de Machine Learning**

**a) Collecte des Données**

Les données sont au cœur du Machine Learning. Pour créer un modèle efficace, il est essentiel de **collecter des données pertinentes** et de qualité. Cela peut provenir de diverses sources (bases de données, capteurs, images, etc.).

**b) Préparation des Données**

Avant d'entrer les données dans un modèle, elles doivent être **nettoyées** et **prétraitées**. Cela peut inclure :

* **Gestion des valeurs manquantes** (imputation ou suppression).
* **Normalisation ou standardisation** des données pour que les algorithmes fonctionnent correctement.
* **Encodage des variables catégorielles** (par exemple, transformer des catégories en nombres).

**c) Choix du Modèle**

Le choix du modèle dépend de la tâche à accomplir (classification, régression, etc.). Certains modèles populaires incluent :

* **Régression Linéaire** pour des prédictions continues.
* **Arbres de Décision** pour des prédictions catégorielles.
* **Réseaux Neuronaux** pour des problèmes plus complexes, comme la reconnaissance d'images.

**d) Entraînement du Modèle**

L'algorithme est **entraîné** sur les données, ce qui consiste à ajuster ses paramètres pour minimiser l'erreur ou maximiser la performance. Cela peut être fait à l'aide de techniques d'**optimisation**, comme la descente de gradient(Ajuster les parametres).

**e) Évaluation du Modèle**

Une fois le modèle entraîné, il est évalué sur un **jeu de données de test** (qui n'a pas été utilisé pendant l'entraînement) pour vérifier sa performance. Les métriques d'évaluation peuvent inclure :

* **Précision**, **Rappel**, **F-score** pour les problèmes de classification.
* **Erreur quadratique moyenne (RMSE)** pour les problèmes de régression.

**f) Amélioration du Modèle**

Si la performance du modèle n'est pas satisfaisante, plusieurs stratégies peuvent être utilisées pour l'améliorer :

* **Ajustement des hyperparamètres** (tuning).
* **Ensemble learning** (combinaison de plusieurs modèles).
* **Augmentation des données** (ajouter plus de données ou utiliser des techniques comme le data augmentation).

**3. Applications du Machine Learning**

Le machine learning est utilisé dans de nombreux domaines :

**a) Santé**

* **Prédiction de maladies** : Prédire si un patient est susceptible de développer une maladie (ex. : cancer).
* **Analyse d'images médicales** : Identifier des anomalies dans les radiographies ou les IRM.

**b) Finance**

* **Détection de fraudes** : Identifier des comportements suspects dans les transactions financières.
* **Prédiction des tendances du marché** : Anticiper les mouvements des actions en bourse.

**c) Marketing**

* **Segmentation de clients** : Regrouper des clients similaires pour des campagnes ciblées.
* **Recommandation de produits** : Systèmes de recommandation comme ceux d'Amazon ou de Netflix.

**d) Transport**

* **Voitures autonomes** : Apprentissage à partir de données de capteurs pour guider les voitures sans intervention humaine.
* **Optimisation des trajets** : Trouver les trajets les plus rapides pour la livraison de marchandises.

**4. Outils et Bibliothèques pour le Machine Learning**

Voici quelques outils populaires utilisés pour le machine learning :

* **Python** : Le langage de programmation le plus utilisé pour le machine learning, avec des bibliothèques comme :
  + **Scikit-learn** : Pour les modèles de machine learning classiques.
  + **TensorFlow** et **PyTorch** : Pour les réseaux neuronaux et l'apprentissage profond.
  + **Keras** : Une bibliothèque de haut niveau pour les réseaux neuronaux.
  + **Pandas** et **NumPy** : Pour la manipulation des données.
* **R** : Un autre langage populaire, surtout pour les statistiques et l'analyse des données.

**Conclusion**

Le machine learning permet de créer des modèles capables d'apprendre et de s'améliorer à partir des données. Il est largement utilisé dans divers secteurs et continue de révolutionner de nombreux domaines. Si vous êtes débutant, commencer par des modèles simples et augmenter progressivement la complexité du problème est un bon point de départ.

* + Différence entre apprentissage supervisé et non-supervisé.

Les concepts d’**apprentissage supervisé** et d’**apprentissage non supervisé** sont fondamentaux en machine learning, et chacun a ses propres caractéristiques et applications. Voici une comparaison détaillée :

### ****1. Apprentissage Supervisé (Supervised Learning)****

#### ****Principe :****

L'apprentissage supervisé consiste à entraîner un modèle à partir de **données étiquetées**. Cela signifie que chaque exemple de données d'entraînement est accompagné d'une **réponse correcte** (une étiquette ou une valeur cible) que l'algorithme doit apprendre à prédire.

#### ****Objectif :****

Le but est de **prédire** ou de **classer** les données futures en utilisant un modèle qui a été formé avec des **entrées** et des **sorties connues**.

#### ****Exemples de tâches :****

* **Classification :** Prédire une étiquette catégorique. Par exemple, déterminer si un e-mail est un **spam** ou non (**spam/non-spam**).
* **Régression :** Prédire une valeur continue. Par exemple, prédire le **prix d'une maison** en fonction de caractéristiques comme la superficie, le nombre de chambres, etc.

#### ****Processus :****

1. **Entraînement** : L'algorithme apprend une fonction qui mappe les données d'entrée à des sorties spécifiques.
2. **Prédiction** : Une fois le modèle formé, il peut prédire la sortie pour de nouvelles données non étiquetées.

#### ****Exemples d'algorithmes :****

* Régression Linéaire
* Arbre de Décision
* Forêt Aléatoire (Random Forest)
* Réseaux de Neurones
* Machines à Vecteurs de Support (SVM)

#### ****Avantages :****

* Précis et efficace lorsque les données étiquetées sont disponibles.
* Le modèle est évalué à l'aide de métriques claires (précision, erreur quadratique moyenne, etc.).

#### ****Inconvénients :****

* Nécessite beaucoup de données étiquetées, ce qui peut être coûteux et long à obtenir.
* Moins flexible face aux nouvelles situations ou données imprévues.

### ****2. Apprentissage Non Supervisé (Unsupervised Learning)****

#### ****Principe :****

L'apprentissage non supervisé fonctionne sur des **données non étiquetées**. L'algorithme cherche à **extraire des structures cachées** ou des **patterns** dans les données sans avoir de réponses connues ou d'étiquettes.

#### ****Objectif :****

Le but est de trouver des **relations ou des structures** au sein des données. L'algorithme n'a pas d'exemple de sortie correcte, et il explore les données par lui-même pour trouver des modèles ou des groupes similaires.

#### ****Exemples de tâches :****

* **Clustering (regroupement) :** Regrouper des données similaires en **clusters**. Par exemple, regrouper des clients en fonction de leur comportement d'achat.
* **Réduction de dimensionnalité :** Réduire le nombre de caractéristiques tout en conservant les informations les plus importantes. Par exemple, réduire un grand nombre de variables dans des images tout en conservant les caractéristiques essentielles pour l'analyse.

#### ****Processus :****

1. **Exploration des données** : L'algorithme analyse les données pour identifier des structures ou des modèles cachés.
2. **Regroupement ou réduction** : Il regroupe les données similaires ou réduit les dimensions pour simplifier l'analyse tout en conservant les caractéristiques clés.

#### ****Exemples d'algorithmes :****

* K-means (pour le clustering)
* Analyse en Composantes Principales (PCA) (pour la réduction de dimensionnalité)
* Clustering hiérarchique
* Auto-encodeurs

#### ****Avantages :****

* Peut fonctionner avec des données non étiquetées, ce qui le rend plus flexible.
* Utilisé pour des tâches exploratoires où il n’y a pas de réponse claire.

#### ****Inconvénients :****

* Les résultats peuvent être moins clairs, car il n’y a pas de **réponse correcte** à évaluer.
* Plus difficile à interpréter, car le modèle peut générer des patterns complexes non évidents.

### ****3. Comparaison Résumée****

| **Caractéristique** | **Apprentissage Supervisé** | **Apprentissage Non Supervisé** |
| --- | --- | --- |
| **Données d'entrée** | Données étiquetées (avec réponse connue) | Données non étiquetées (pas de réponse connue) |
| **Objectif** | Prédiction ou classification (prédire une étiquette ou une valeur) | Découvrir des structures cachées ou des groupes dans les données |
| **Exemples de tâches** | Classification, Régression | Clustering, Réduction de dimensionnalité |
| **Exemples d'algorithmes** | Régression linéaire, Arbres de décision, Réseaux neuronaux, SVM | K-means, PCA, Clustering hiérarchique, Auto-encodeurs |
| **Étiquettes de données** | Requises pour l'apprentissage | Non requises |
| **Évaluation du modèle** | Évalué par des métriques comme la précision, l'erreur quadratique | Pas d'évaluation directe (évalué par la qualité des clusters ou des patterns découverts) |
| **Complexité** | Plus simple, nécessite des données étiquetées | Plus complexe, nécessite d’interpréter les modèles générés |

### ****4. Applications Pratiques****

* **Apprentissage Supervisé :**
  + **Classification** : Reconnaissance de lettres dans un CAPTCHA, reconnaissance faciale, diagnostic médical basé sur des symptômes.
  + **Régression** : Prédiction des prix immobiliers, estimation des ventes futures d'un produit.
* **Apprentissage Non Supervisé :**
  + **Clustering** : Segmentation de marché pour cibler différents groupes de consommateurs, regroupement d'articles similaires dans des systèmes de recommandation.
  + **Réduction de dimensionnalité** : Compression d’images, réduction du bruit dans les données, simplification des jeux de données complexes.

### ****Conclusion :****

* **Apprentissage supervisé** : Utilisé lorsqu’on connaît les étiquettes de données et que l’on cherche à faire des prédictions sur de nouvelles données.
* **Apprentissage non supervisé** : Utilisé pour explorer des données sans étiquettes et découvrir des structures cachées ou des groupes dans les données.
  + **Jeu de données d'entraînement et de test.**

Dans le cadre du **Machine Learning**, il est courant de diviser les données en plusieurs ensembles, notamment un **jeu de données d'entraînement** et un **jeu de données de test**. Ces deux jeux de données servent à différentes étapes du processus de modélisation et permettent d'évaluer la performance du modèle de manière robuste.

### ****1. Jeu de Données d'Entraînement (Training Set)****

#### ****Définition :****

Le **jeu de données d'entraînement** est utilisé pour **former** (ou entraîner) le modèle. C’est avec ces données que l'algorithme apprend la relation entre les **entrées** (features) et les **sorties** (étiquettes ou valeurs cibles).

#### ****Rôle :****

* **Entraîner le modèle** : L'algorithme ajuste ses paramètres en fonction des données d'entraînement.
* **Apprentissage des patterns** : C'est à partir de ces données que le modèle apprend à prédire ou à classifier des données inconnues.

#### ****Caractéristiques :****

* Le modèle **voit** ces données et ajuste ses paramètres internes pour minimiser l'erreur.
* L'algorithme peut **mémoriser** les exemples d'entraînement et **s'adapter** à eux. Cependant, cela peut entraîner un **surapprentissage (overfitting)** si le modèle est trop complexe.

#### ****Taille :****

* Généralement, un **pourcentage élevé** des données disponibles (par exemple, 70-80%) est utilisé pour l'entraînement.

### ****2. Jeu de Données de Test (Test Set)****

#### ****Définition :****

Le **jeu de données de test** est un ensemble de données **distinct** du jeu d'entraînement, qui est utilisé pour **évaluer la performance** du modèle une fois qu'il a été formé.

#### ****Rôle :****

* **Évaluation de la performance** : Le modèle n'a pas vu ces données pendant l'entraînement. On les utilise pour évaluer à quel point le modèle généralise bien à de nouvelles données.
* **Mesure de l'overfitting** : Si un modèle donne de très bons résultats sur l'ensemble d'entraînement mais de mauvais résultats sur l'ensemble de test, cela peut indiquer un surapprentissage, où le modèle a **mémorisé** les données d'entraînement au lieu d'apprendre des patterns généraux.

#### ****Caractéristiques :****

* Ces données ne sont **jamais** utilisées pendant la phase d'entraînement.
* Elles servent uniquement à **tester** le modèle sur de nouvelles données, qu'il n'a jamais rencontrées.

#### ****Taille :****

* En général, environ **20-30%** des données sont réservées pour le jeu de test.

### ****3. Pourquoi Utiliser un Jeu de Données d'Entraînement et un Jeu de Données de Test ?****

#### ****a) Éviter l'Overfitting****

* Si un modèle est uniquement évalué sur les données qu’il a vues pendant l'entraînement, il pourrait **s'adapter trop étroitement** à ces données, capturant même le bruit ou les détails inutiles. Cela signifie qu'il pourrait échouer à généraliser sur de nouvelles données (inconnues).
* En divisant les données en **jeu d'entraînement** et **jeu de test**, on s'assure que le modèle n'est pas seulement bon sur les données qu'il a vues, mais qu'il peut également bien performer sur de nouvelles données.

#### ****b) Mesurer la Performance Réelle****

* Le **jeu de test** est utilisé pour estimer la **vraie performance** du modèle dans un environnement réel. C’est comme tester une voiture sur une route après l’avoir assemblée — on veut savoir comment elle réagit dans des conditions réelles, et non dans un cadre contrôlé.

#### ****c) Validation du Modèle****

* Les **données de test** permettent de valider l'hypothèse selon laquelle le modèle pourra bien se généraliser à de nouvelles données. Si le modèle fonctionne bien sur les deux jeux de données (entraînement et test), il est plus probable qu’il puisse bien fonctionner dans des scénarios réels.

### ****4. Validation Croisée (Cross-Validation)****

Une autre technique populaire, surtout lorsque les données disponibles sont limitées, est la **validation croisée (cross-validation)**. Au lieu de simplement séparer les données en un seul jeu d'entraînement et un jeu de test, la validation croisée divise les données en plusieurs sous-ensembles appelés **folds**.

#### ****Principe de la validation croisée :****

1. Les données sont divisées en plusieurs sous-ensembles égaux (par exemple, 5 ou 10 **folds**).
2. À chaque itération, un fold différent est utilisé comme **jeu de test**, et les autres sont utilisés pour entraîner le modèle.
3. Le modèle est évalué à chaque itération, et la performance moyenne est calculée.

Cela permet d'utiliser toutes les données à la fois pour l'entraînement et pour le test, réduisant ainsi le biais d'évaluation et offrant une estimation plus fiable de la performance du modèle.

### ****5. Exemple Concret en Python avec Scikit-learn****

Voici un exemple simple montrant comment diviser un jeu de données en un jeu d'entraînement et un jeu de test avec **Scikit-learn** :

*from sklearn.model\_selection import train\_test\_split*

*import numpy as np*

*# Exemple de données fictives*

*X = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]) # Features*

*y = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) # Target*

*# Séparer les données en jeu d'entraînement et jeu de test (80% entraînement, 20% test)*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*print("Données d'entraînement (X\_train) :", X\_train)*

*print("Données de test (X\_test) :", X\_test)*

*print("Cibles d'entraînement (y\_train) :", y\_train)*

*print("Cibles de test (y\_test) :", y\_test)*

Dans cet exemple :

* **train\_test\_split** permet de diviser les données en un jeu d'entraînement et un jeu de test. Le paramètre test\_size=0.2 signifie que 20% des données sont réservées pour le test et 80% pour l'entraînement.
* Le **paramètre random\_state** garantit que la séparation est reproductible à chaque exécution.

### ****6. Conclusion****

* Le **jeu d'entraînement** est utilisé pour entraîner le modèle, tandis que le **jeu de test** est utilisé pour évaluer sa performance.
* Cette division permet de vérifier la capacité du modèle à **généraliser** et à bien performer sur des données qu’il n'a pas vues pendant l'entraînement.
* La **validation croisée** est une autre approche permettant d’évaluer plus précisément la performance du modèle lorsque les données sont limitées.
  + **Sélection des variables pertinentes.**

La **sélection des variables pertinentes**, ou **sélection des caractéristiques (features)**, est une étape cruciale dans le processus de machine learning. Elle consiste à **choisir** les **variables** les plus **informatives** et les plus **utiles** pour la construction du modèle, tout en **éliminant celles qui sont redondantes**, **non informatives** ou **bruyantes**.

La sélection des variables peut améliorer la **performance** du modèle, le rendre **plus rapide** et **réduire le risque de surapprentissage (overfitting)**. Elle permet également de **simplifier** le modèle et de le rendre plus facile à interpréter.

### ****1. Pourquoi effectuer une sélection de variables ?****

#### ****a) Amélioration de la performance du modèle****

* En éliminant des variables non pertinentes, on réduit la **complexité** du modèle, ce qui peut améliorer sa **précision** et **généralisation**.
* Un modèle avec trop de variables peut sur-apprendre les données d'entraînement (overfitting), ce qui nuit à sa capacité à généraliser sur des données nouvelles.

#### ****b) Réduction du temps de calcul****

* Moins de variables impliquent des calculs plus rapides et moins de **données** à traiter pendant l'entraînement et la prédiction.

#### ****c) Simplification et interprétabilité****

* Un modèle avec moins de variables est souvent plus facile à comprendre et à interpréter. Cela permet de mieux comprendre les relations entre les variables et la cible (variable à prédire).

#### ****d) Amélioration de la précision du modèle****

* Parfois, des variables inutiles peuvent introduire du **bruit**, ce qui diminue la précision du modèle. En éliminant ces variables, le modèle peut se concentrer sur les aspects les plus pertinents des données.

### ****2. Méthodes de Sélection des Variables****

Il existe plusieurs approches pour effectuer la sélection des variables, qui peuvent être divisées en trois grandes catégories : **méthodes filtrantes**, **méthodes wrapper**, et **méthodes intégrées**.

#### ****a) Méthodes Filtrantes (Filter Methods)****

Les **méthodes filtrantes** évaluent chaque variable individuellement en fonction de **critères statistiques** ou de **corrélation** et sélectionnent les plus pertinentes avant de les passer dans le modèle.

* **Critères statistiques courants :**
  + **Corrélation** : Vérifier la corrélation entre les variables et la cible. Les variables avec une faible corrélation peuvent être éliminées.
  + **Tests statistiques** : Par exemple, le test de **Chi-carré** pour les variables catégorielles ou le test de **t-Student** pour les variables continues.
  + **Information mutuelle (Mutual Information)** : Mesure la dépendance entre deux variables.

**Exemple avec Python (corrélation) :**

*import pandas as pd*

*import seaborn as sns*

*# Chargement des données*

*data = pd.read\_csv('data.csv')*

*# Matrice de corrélation*

*correlation\_matrix = data.corr()*

*# Visualisation de la matrice de corrélation*

*sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm")*

#### ****b) Méthodes Wrapper (Wrapper Methods)****

Les **méthodes wrapper** utilisent un modèle d'apprentissage pour **évaluer les performances** des différentes combinaisons de variables. Ces méthodes sont plus coûteuses en termes de temps de calcul, mais elles tendent à donner de meilleurs résultats en termes de précision.

* **Méthodes courantes :**
  + **Sélection par recherche exhaustive** : Tester toutes les combinaisons possibles de variables (très coûteux en temps).
  + **Sélection par recherche en avant (Forward Selection)** : Commence avec un modèle sans variables et ajoute les variables une par une en fonction de l'amélioration de la performance.
  + **Sélection par recherche en arrière (Backward Elimination)** : Commence avec toutes les variables et élimine les moins pertinentes une par une.
  + **Sélection par recherche bidirectionnelle** : Combine les deux stratégies précédentes.

**Exemple avec Python (sélection par recherche en avant avec RFE - Recursive Feature Elimination):**

*from sklearn.feature\_selection import RFE*

*from sklearn.linear\_model import LogisticRegression*

*# Définir le modèle*

*model = LogisticRegression()*

*# Appliquer RFE pour sélectionner les variables*

*selector = RFE(model, n\_features\_to\_select=5) # Sélectionner les 5 meilleures variables*

*X\_selected = selector.fit\_transform(X\_train, y\_train)*

*# Afficher les variables sélectionnées*

*selected\_features = X\_train.columns[selector.support\_]*

*print("Variables sélectionnées : ", selected\_features)*

#### ****c) Méthodes Intégrées (Embedded Methods)****

Les **méthodes intégrées** sélectionnent les variables pendant l'apprentissage du modèle. Ces méthodes intègrent la sélection des variables dans le processus d'entraînement et sont généralement plus rapides que les méthodes wrapper.

* **Exemples d'algorithmes avec sélection intégrée des variables :**
  + **Forêts aléatoires (Random Forest)** : Utilise l'importance des variables pour sélectionner les meilleures caractéristiques.
  + **Lasso (L1 Regularization)** : L'ajout de régularisation L1 dans un modèle de régression linéaire ou logistique permet de forcer certains coefficients de variables à zéro, effectuant ainsi une sélection de variables.

**Exemple avec Python (Lasso pour sélection de variables) :**

*from sklearn.linear\_model import Lasso*

*from sklearn.preprocessing import StandardScaler*

*# Normalisation des données*

*scaler = StandardScaler()*

*X\_scaled = scaler.fit\_transform(X\_train)*

*# Appliquer Lasso*

*lasso = Lasso(alpha=0.01) # Valeur de alpha pour régulariser*

*lasso.fit(X\_scaled, y\_train)*

*# Afficher les coefficients*

*coefficients = lasso.coef\_*

*selected\_features = X\_train.columns[coefficients != 0]*

*print("Variables sélectionnées par Lasso : ", selected\_features)*

### ****3. Critères pour Choisir une Méthode de Sélection****

* **Nombre de Variables** : Si vous avez un grand nombre de variables, une méthode comme **RFE** ou l’utilisation de modèles comme **Random Forest** peut être plus efficace.
* **Coût Computationnel** : Les **méthodes wrapper** peuvent être très coûteuses en termes de calculs, surtout si vous travaillez avec de grandes quantités de données. Les **méthodes filtrantes** sont plus simples et rapides.
* **Type de Données** : Si vos données sont principalement numériques, les **méthodes filtrantes** (comme la corrélation) peuvent suffire. Pour des données complexes ou non linéaires, les **méthodes wrapper** ou **intégrées** peuvent mieux performer.

### ****4. Exemples Pratiques de Sélection de Variables****

* **Exemple 1 :** Si vous avez un modèle de régression avec 100 variables, vous pouvez utiliser **Lasso** pour réduire automatiquement les variables inutiles en appliquant une régularisation L1.
* **Exemple 2 :** Si vous travaillez avec un modèle comme **Random Forest**, vous pouvez utiliser les **importances des variables** pour sélectionner les variables les plus influentes.

### ****5. Conclusion****

La sélection des variables pertinentes est un aspect clé du machine learning, car elle influence directement la performance du modèle, son interprétabilité et le temps de calcul. Il existe différentes techniques de sélection des variables, et le choix de la méthode dépend du type de données, du modèle utilisé et des ressources disponibles. Une bonne sélection des variables peut améliorer de manière significative la capacité du modèle à généraliser et à fournir des prédictions précises sur de nouvelles données.

1. **Régression et Classification**

La **régression** et la **classification** sont deux des techniques les plus courantes en **Machine Learning supervisé**. Elles sont utilisées pour prédire une variable cible (ou **variable à prédire**) en fonction d'un ensemble de variables explicatives (**features** ou **entrées**).

La principale différence entre ces deux types de problèmes réside dans la **nature de la variable cible** (la variable que l'on cherche à prédire) :

* **Régression** : La variable cible est **continue**.
* **Classification** : La variable cible est **discrète** ou **catégorique**.

### ****1. Régression****

#### ****Définition :****

La **régression** est utilisée pour prédire une variable **continue** (numérique) en fonction d'un ou plusieurs **predictors** (caractéristiques). L'objectif est de trouver une relation entre les variables explicatives et la variable à prédire.

#### ****Exemples de problèmes de régression :****

* Prédire le **prix** d'une maison en fonction de ses caractéristiques (superficie, nombre de chambres, emplacement, etc.).
* Estimer la **température** demain en fonction des conditions météorologiques actuelles.
* Prédire les **ventes** d'un produit en fonction de la publicité et des promotions.

#### ****Modèles de régression courants :****

1. **Régression linéaire** : La relation entre les variables est modélisée par une ligne droite.
2. **Régression polynomiale** : Extension de la régression linéaire où la relation est modélisée par un polynôme.
3. **Régression Ridge/Lasso** : Régressions linéaires avec régularisation pour éviter le surapprentissage.
4. **Régression logistique** (bien qu'utilisée pour la classification, elle est souvent utilisée pour des probabilités dans des problèmes de régression avec une variable cible binaire).

#### ****Exemple avec la régression linéaire en Python :****

*import numpy as np*

*import pandas as pd*

*from sklearn.linear\_model import LinearRegression*

*from sklearn.model\_selection import train\_test\_split*

*# Exemple de données*

*X = np.array([[1], [2], [3], [4], [5]]) # Taille des maisons (en m²)*

*y = np.array([100, 150, 200, 250, 300]) # Prix des maisons (en milliers d'euros)*

*# Séparation des données en jeu d'entraînement et jeu de test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Initialisation du modèle*

*model = LinearRegression()*

*# Entraînement du modèle*

*model.fit(X\_train, y\_train)*

*# Prédiction sur le jeu de test*

*y\_pred = model.predict(X\_test)*

*print("Prédictions :", y\_pred)*

### ****2. Classification****

#### ****Définition :****

La **classification** est utilisée pour prédire une variable **catégorique**. L'objectif est de classer les entrées dans une ou plusieurs catégories (ou classes).

#### ****Exemples de problèmes de classification :****

* Classer des emails en **spam** ou **non-spam**.
* Diagnostiquer si un patient a une **maladie** ou non (binaire : oui/non).
* Catégoriser des images de **fruits** en fonction de leur type (par exemple, pomme, banane, orange).

#### ****Modèles de classification courants :****

1. **Régression logistique** : Utilisée pour des problèmes de classification binaire (deux classes), bien qu'on l'appelle "régression", elle sert à classifier.
2. **Arbres de décision** : Divise les données en fonction de questions sur les caractéristiques pour classer les observations.
3. **Forêts aléatoires (Random Forest)** : Un ensemble d'arbres de décision, souvent plus précis.
4. **Machines à vecteurs de support (SVM)** : Trouve la meilleure frontière (hyperplan) pour séparer les différentes classes.
5. **k-NN (k-plus proches voisins)** : Classe les observations en fonction de la majorité des voisins proches.
6. **Réseaux de neurones** : Utilisés dans des problèmes complexes, notamment pour la reconnaissance d'images et le traitement du langage naturel.

#### ****Exemple avec la régression logistique en Python :****

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.datasets import load\_iris*

*from sklearn.model\_selection import train\_test\_split*

*# Charger le jeu de données Iris*

*data = load\_iris()*

*X = data.data*

*y = data.target*

*# Séparation des données en jeu d'entraînement et jeu de test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)*

*# Initialisation du modèle*

*model = LogisticRegression(max\_iter=200)*

*# Entraînement du modèle*

*model.fit(X\_train, y\_train)*

*# Prédiction sur le jeu de test*

*y\_pred = model.predict(X\_test)*

*print("Prédictions :", y\_pred)*

### ****3. Comparaison entre Régression et Classification****

| **Critère** | **Régression** | **Classification** |
| --- | --- | --- |
| **Type de variable cible** | Continue (numérique) | Discrète (catégorique) |
| **Objectif** | Prédire une valeur numérique | Attribuer des observations à des classes |
| **Exemples de modèles** | Régression linéaire, Ridge, Lasso | Régression logistique, SVM, k-NN, Random Forest |
| **Évaluation de la performance** | Erreur quadratique moyenne (MSE), MAE | Précision, rappel, F1-score, matrice de confusion |
| **Utilisation** | Estimation de quantités continues (prix, température, etc.) | Catégorisation (spam/non-spam, maladie/non-maladie) |

### ****4. Conclusion****

La **régression** et la **classification** sont les deux piliers des problèmes supervisés en machine learning. La principale différence réside dans la nature de la variable cible :

* En **régression**, la variable cible est **continue**, et l’objectif est de prédire une quantité numérique.
* En **classification**, la variable cible est **discrète**, et l’objectif est de prédire des classes ou des catégories.

Les deux types d'apprentissage supervisé utilisent des techniques différentes, mais l'idée fondamentale reste la même : **apprendre à partir des données étiquetées** pour faire des prédictions sur de nouvelles données.

* + **Régression linéaire et logistique**.

La **régression linéaire** et la **régression logistique** sont deux modèles fondamentaux en **machine learning supervisé**, mais elles sont utilisées dans des contextes différents en raison des différences dans la nature de la variable cible et des objectifs des modèles.

#### ****1. Régression Linéaire****

**La régression linéaire** est une méthode utilisée pour prédire une **variable continue** à partir d'une ou plusieurs variables explicatives. Elle suppose qu'il existe une relation **linéaire** entre la variable cible et les variables explicatives.

#### ****Principe :****

L’objectif de la régression linéaire est de modéliser la relation entre la variable cible yyy et les variables explicatives XXX à l’aide d’une équation linéaire.

#### ****Exemple d'application :****

* **Prédire le prix d'une maison** en fonction de caractéristiques comme la superficie, le nombre de chambres, etc.
* **Estimer la température** en fonction de l'heure de la journée, de la saison, etc.

#### ****Évaluation du modèle de régression linéaire :****

Les modèles de régression linéaire sont évalués en fonction de l'erreur entre les valeurs réelles et les valeurs prédites. Les critères courants incluent :

* **Erreur quadratique moyenne (MSE - Mean Squared Error)**
* **R² (coefficient de détermination)** : Mesure la proportion de la variance de la variable cible qui est expliquée par le modèle.

#### ****Exemple de régression linéaire en Python :****

from sklearn.linear\_model import LinearRegression

import numpy as np

# Exemple de données

X = np.array([[1], [2], [3], [4], [5]]) # Taille des maisons en m²

y = np.array([100, 150, 200, 250, 300]) # Prix des maisons en milliers d'euros

# Créer et entraîner le modèle

model = LinearRegression()

model.fit(X, y)

# Faire des prédictions

y\_pred = model.predict([[6]]) # Prédire le prix pour une maison de 6 m²

print("Prix prédit :", y\_pred)

#### ****2. Régression Logistique****

**La régression logistique** est un modèle utilisé pour résoudre des problèmes de **classification binaire**. Contrairement à la régression linéaire, qui prédit des valeurs continues, la régression logistique prédit la **probabilité** qu'une observation appartienne à une classe donnée (par exemple, classe 1 ou classe 0).

#### ****Principe :****

La régression logistique utilise une fonction **sigmoïde (ou logistique)** pour modéliser la relation entre les variables explicatives et la probabilité de l'appartenance à une classe.

#### ****Exemple d'application :****

* **Prédire si un email est un spam ou non (0 ou 1)**.
* **Diagnostiquer une maladie** : Si une personne a la maladie (1) ou non (0), en fonction des caractéristiques de santé.

#### ****Évaluation du modèle de régression logistique :****

Les modèles de régression logistique sont souvent évalués à l’aide des critères suivants :

* **Accuracy (Précision)** : Proportion de prédictions correctes.
* **Matrice de confusion** : Un tableau qui présente les vrais positifs, faux positifs, vrais négatifs et faux négatifs.
* **AUC-ROC (Area Under the Curve - Receiver Operating Characteristic)** : Évalue la performance du modèle en termes de ses capacités à distinguer les classes.

#### ****Exemple de régression logistique en Python :****

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.datasets import load\_iris*

*from sklearn.model\_selection import train\_test\_split*

*# Charger les données Iris (on choisit les 2 premières classes pour classification binaire)*

*data = load\_iris()*

*X = data.data[data.target != 2] # On ne garde que les classes 0 et 1*

*y = data.target[data.target != 2]*

*# Séparation en données d'entraînement et de test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)*

*# Créer et entraîner le modèle*

*model = LogisticRegression(max\_iter=200)*

*model.fit(X\_train, y\_train)*

*# Faire des prédictions*

*y\_pred = model.predict(X\_test)*

*print("Prédictions :", y\_pred)*

### ****3. Différences entre Régression Linéaire et Régression Logistique****

| **Critère** | **Régression Linéaire** | **Régression Logistique** |
| --- | --- | --- |
| **Type de variable cible** | Continue (ex : prix, température) | Discrète (binaire : 0 ou 1) |
| **Type de modèle** | Prédit une valeur numérique (quantité) | Prédit une probabilité (appartenance à une classe) |
| **Fonction utilisée** | Fonction linéaire | Fonction sigmoïde (logistique) |
| **Sortie** | Valeur continue | Probabilité (comprise entre 0 et 1) |
| **Utilisation** | Estimation des quantités continues | Classification binaire (ex : spam/non-spam) |
| **Évaluation** | Erreur quadratique moyenne (MSE), R² | Précision, rappel, matrice de confusion, AUC |

### ****4. Conclusion****

* **Régression linéaire** : Utilisée pour prédire une **variable continue** en fonction des variables explicatives.
* **Régression logistique** : Utilisée pour la **classification binaire**, afin de prédire la probabilité d'appartenance à une classe.

Chacune de ces régressions a son propre domaine d'application et peut être choisie en fonction du type de problème (régression ou classification) et du type de variable cible.

* + Algorithmes de classification : **k-Nearest Neighbors (KNN)**, **Random Forest**, **SVM**.

Les algorithmes de classification sont utilisés pour prédire une catégorie ou une classe à partir de données d'entrée. Parmi les plus populaires, on trouve **k-Nearest Neighbors (KNN)**, **Random Forest** et **Support Vector Machine (SVM)**. Chaque algorithme a ses propres caractéristiques, avantages et inconvénients.

### ****1. k-Nearest Neighbors (KNN)****

#### ****Principe :****

Le **k-Nearest Neighbors (KNN)** est un algorithme de classification basé sur la proximité des points de données. Il fonctionne de la manière suivante :

1. Pour chaque nouvelle observation à classer, l'algorithme cherche les **k plus proches voisins** dans l'ensemble d'apprentissage.
2. La classe de l'observation est déterminée par **majorité** parmi les classes des k voisins les plus proches.

#### ****Calcul de la proximité :****

La proximité est généralement mesurée à l'aide d'une **distance** entre les points. Les plus courantes sont :

* **Distance Euclidienne**
* **Distance de Manhattan** **Choix de k :**
* Si **k = 1**, chaque point est classé par rapport à son voisin le plus proche.
* Un **k** plus grand peut aider à éviter les fluctuations dues aux points aberrants, mais un **k** trop grand peut rendre le modèle moins sensible aux variations locales.

#### ****Exemple d'application :****

* Classification d'images en fonction des caractéristiques de l'image.
* Prédiction de la catégorie d'un email (spam/non-spam) en fonction de ses mots-clés.

#### ****Exemple de KNN en Python :****

*from sklearn.neighbors import KNeighborsClassifier*

*from sklearn.datasets import load\_iris*

*from sklearn.model\_selection import train\_test\_split*

*# Charger le jeu de données Iris*

*data = load\_iris()*

*X = data.data*

*y = data.target*

*# Séparer les données en jeu d'entraînement et de test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)*

*# Créer le modèle KNN avec k=3*

*model = KNeighborsClassifier(n\_neighbors=3)*

*# Entraîner le modèle*

*model.fit(X\_train, y\_train)*

*# Faire des prédictions*

*y\_pred = model.predict(X\_test)*

*print("Prédictions :", y\_pred)*

### ****2. Random Forest****

#### ****Principe :****

**Random Forest** est un algorithme d'**ensemble learning** basé sur la construction de plusieurs **arbres de décision** et leur agrégation pour produire une prédiction plus robuste.

1. **Création de multiples arbres de décision** : Chaque arbre est construit sur un sous-ensemble aléatoire des données d'entraînement (avec remplacement) et utilise un sous-ensemble aléatoire des caractéristiques à chaque division.
2. **Prédiction** : La prédiction finale est obtenue par **vote majoritaire** pour les problèmes de classification.

#### ****Avantages :****

* Moins susceptible au surapprentissage (overfitting) par rapport à un seul arbre de décision.
* Peut capturer des relations complexes entre les données.
* Peut gérer des données manquantes et des variables non-linéaires.

#### ****Exemple d'application :****

* Prévisions de crédit (ex : approuver ou refuser un prêt).
* Classification de textes ou d'images.

#### ****Exemple de Random Forest en Python :****

*from sklearn.ensemble import RandomForestClassifier*

*from sklearn.datasets import load\_iris*

*from sklearn.model\_selection import train\_test\_split*

*# Charger le jeu de données Iris*

*data = load\_iris()*

*X = data.data*

*y = data.target*

*# Séparer les données en jeu d'entraînement et de test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)*

*# Créer le modèle Random Forest*

*model = RandomForestClassifier(n\_estimators=100)*

*# Entraîner le modèle*

*model.fit(X\_train, y\_train)*

*# Faire des prédictions*

*y\_pred = model.predict(X\_test)*

*print("Prédictions :", y\_pred)*

### ****3. Support Vector Machine (SVM)****

#### ****Principe :****

Les **Support Vector Machines (SVM)** sont des algorithmes de classification qui trouvent la **frontière optimale** entre différentes classes en maximisant la **marge** entre les classes. L'idée est de séparer les classes en traçant un hyperplan qui maximise la distance entre les points les plus proches de chaque classe (les **vecteurs de support**).

1. **Séparation linéaire** : Si les données sont linéairement séparables, l'SVM cherche l'hyperplan qui sépare les classes avec la plus grande marge.
2. **Séparation non linéaire** : Si les données ne sont pas linéairement séparables, l'SVM utilise un **noyau (kernel)** pour projeter les données dans un espace de dimension plus élevée où elles peuvent être séparées linéairement.

#### ****Types de noyaux courants :****

* **Noyau linéaire** : Utilisé lorsque les données sont linéairement séparables.
* **Noyau RBF (Radial Basis Function)** : Utilisé pour des données non linéaires.
* **Noyau polynomial** : Pour des relations non linéaires complexes.

#### ****Exemple d'application :****

* Classification de textes (ex : analyse de sentiments, spam/non-spam).
* Classification d'images et reconnaissance de formes.

#### ****Exemple de SVM en Python :****

*from sklearn.svm import SVC*

*from sklearn.datasets import load\_iris*

*from sklearn.model\_selection import train\_test\_split*

*# Charger le jeu de données Iris*

*data = load\_iris()*

*X = data.data*

*y = data.target*

*# Séparer les données en jeu d'entraînement et de test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)*

*# Créer le modèle SVM avec noyau RBF*

*model = SVC(kernel='rbf')*

*# Entraîner le modèle*

*model.fit(X\_train, y\_train)*

*# Faire des prédictions*

*y\_pred = model.predict(X\_test)*

*print("Prédictions :", y\_pred)*

### ****Comparaison des Algorithmes de Classification****

| **Critère** | **k-Nearest Neighbors (KNN)** | **Random Forest** | **Support Vector Machine (SVM)** |
| --- | --- | --- | --- |
| **Principe** | Classifie en fonction des voisins les plus proches | Ensemble d'arbres de décision, vote majoritaire | Trouve la frontière optimale pour séparer les classes |
| **Type de données** | Convient pour les données avec des frontières simples | Convient pour les données complexes et non linéaires | Convient pour les données linéairement et non linéairement séparables |
| **Vitesse d'entraînement** | Lente (doit calculer les distances pour chaque point de test) | Rapide à l'entraînement, surtout avec un grand nombre d'arbres | Relativement lent à l'entraînement, surtout pour de grandes quantités de données |
| **Vitesse de prédiction** | Lente (calcul de distance pour chaque prédiction) | Rapide, car chaque arbre donne une prédiction indépendante | Rapide pour les petites données, mais peut être lent avec de grandes bases |
| **Gestion des données manquantes** | Ne gère pas bien les données manquantes | Gère bien les données manquantes | Ne gère pas bien les données manquantes sans prétraitement |
| **Sensibilité aux données aberrantes** | Très sensible aux outliers | Moins sensible aux outliers | Sensible aux outliers mais peut être ajusté avec des paramètres |
| **Facilité d'interprétation** | Simple à comprendre et à expliquer | Moins interprétable en raison de l'ensemble d'arbres | Moins interprétable, mais efficace pour des tâches complexes |
| **Utilisation courante** | Classification de petites à moyennes bases de données | Classification de données complexes et non linéaires | Classification complexe avec des marges claires |

### ****Conclusion****

* **KNN** est simple à comprendre et efficace pour des ensembles de données de petite taille, mais peut devenir lent pour de grandes bases de données.
* **Random Forest** est robuste, puissant et efficace pour gérer des données complexes avec un grand nombre de variables.
* **SVM** est très performant pour les tâches de classification complexes, en particulier lorsque les classes sont bien séparées, mais il peut être plus lent à entraîner.

Chacun de ces algorithmes a ses avantages et inconvénients en fonction des données et des besoins du problème de classification.

* + Courbe ROC et métriques d’évaluation.

Lorsqu'on entraîne un modèle de classification, il est crucial d’évaluer sa performance à l’aide de **métriques d’évaluation** adaptées. Parmi les plus utilisées, on retrouve **l’accuracy, la précision, le rappel, le F1-score** et **la courbe ROC/AUC**.

## **1. Les principales métriques d'évaluation en classification**

### ****1.1 Matrice de confusion****

La **matrice de confusion** est un tableau qui compare les **prédictions** d'un modèle avec les **valeurs réelles**. Elle est structurée comme suit pour une classification binaire :

|  | **Prédit : Positif (1)** | **Prédit : Négatif (0)** |
| --- | --- | --- |
| **Réel : Positif (1)** | Vrai Positif (VP) | Faux Négatif (FN) |
| **Réel : Négatif (0)** | Faux Positif (FP) | Vrai Négatif (VN) |

* **Vrai Positif (VP)** : L'exemple était positif et le modèle l'a correctement classé.
* **Faux Négatif (FN)** : L'exemple était positif mais le modèle l'a mal classé comme négatif.
* **Faux Positif (FP)** : L'exemple était négatif mais le modèle l'a mal classé comme positif.
* **Vrai Négatif (VN)** : L'exemple était négatif et le modèle l'a correctement classé.

### ****1.2 Accuracy (Taux de bonnes classifications)****

L’**accuracy** mesure le pourcentage de bonnes prédictions sur l’ensemble des observations.

**Avantage** : Facile à comprendre.  
**Inconvénient** : Peut être trompeur en cas de classes déséquilibrées (ex : 95% de non-malades et 5% de malades).

### ****1.3 Précision (Precision)****

La **précision** mesure la proportion de **vrais positifs** parmi tous les **prédits positifs**.

Elle indique à quel point le modèle **évite les fausses alarmes** (FP).

* Utile dans les contextes où **les faux positifs sont coûteux** (ex : détection de fraudes bancaires).

### ****1.4 Rappel (Recall ou Sensibilité)****

Le **rappel** mesure la proportion de **vrais positifs** correctement détectés parmi tous les **réels positifs**.

Il montre si le modèle détecte **tous les cas positifs** (FN faibles).

* Important quand **les faux négatifs sont critiques** (ex : diagnostic médical).

### ****1.5 F1-Score (Compromis entre Précision et Rappel)****

L'**F1-score** est la moyenne harmonique de la **précision** et du **rappel** :

* **Si F1-score proche de 1** → Bon équilibre entre précision et rappel.
* **Si F1-score proche de 0** → Mauvaise performance du modèle.

## **2. Courbe ROC et AUC (Area Under Curve)**

### ****2.1 Qu'est-ce que la courbe ROC ?****

La **courbe ROC (Receiver Operating Characteristic)** est un graphique qui montre la performance d’un modèle de classification en fonction de **différents seuils de décision**.

Elle trace le **taux de vrais positifs (rappel)** en fonction du **taux de faux positifs** :

* **Axe X** : Faux positifs **(1 - Spécificité)**
* **Axe Y** : Vrais positifs **(Rappel)**

Un **modèle parfait** serait représenté par une courbe ROC qui monte verticalement jusqu'à 1, puis reste horizontale.

### ****2.2 AUC (Area Under the Curve)****

L’**AUC** est l'aire sous la courbe ROC. C'est un indicateur qui mesure la capacité du modèle à **séparer correctement les classes** :

| **Valeur de l’AUC** | **Interprétation** |
| --- | --- |
| **0.5** | Performance équivalente au hasard (mauvais modèle). |
| **0.7 - 0.8** | Bonne capacité de classification. |
| **0.8 - 0.9** | Très bon modèle. |
| **0.9 - 1** | Excellent modèle. |

**Exemple** : Si AUC = 0.9, cela signifie que dans **90% des cas**, le modèle classe correctement un positif avant un négatif.

### ****3. Implémentation en Python****

Voyons comment utiliser ces métriques en **Python** avec scikit-learn sur un modèle **Random Forest**.

*from sklearn.ensemble import RandomForestClassifier*

*from sklearn.datasets import load\_iris*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, roc\_curve, auc*

*import matplotlib.pyplot as plt*

*# Charger les données Iris*

*data = load\_iris()*

*X = data.data*

*y = (data.target == 2).astype(int) # On transforme en problème binaire (classe 2 vs reste)*

*# Séparer les données en train/test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)*

*# Entraîner un modèle Random Forest*

*model = RandomForestClassifier(n\_estimators=100)*

*model.fit(X\_train, y\_train)*

*# Prédire sur le test*

*y\_pred = model.predict(X\_test)*

*y\_prob = model.predict\_proba(X\_test)[:, 1] # Probabilité d'appartenir à la classe 1*

*# Calcul des métriques*

*accuracy = accuracy\_score(y\_test, y\_pred)*

*precision = precision\_score(y\_test, y\_pred)*

*recall = recall\_score(y\_test, y\_pred)*

*f1 = f1\_score(y\_test, y\_pred)*

*conf\_matrix = confusion\_matrix(y\_test, y\_pred)*

*# Afficher la matrice de confusion*

*print("Matrice de confusion :\n", conf\_matrix)*

*print(f"Accuracy : {accuracy:.2f}")*

*print(f"Précision : {precision:.2f}")*

*print(f"Rappel : {recall:.2f}")*

*print(f"F1-Score : {f1:.2f}")*

*# Courbe ROC*

*fpr, tpr, \_ = roc\_curve(y\_test, y\_prob)*

*roc\_auc = auc(fpr, tpr)*

*plt.figure(figsize=(8,6))*

*plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc\_auc:.2f})')*

*plt.plot([0, 1], [0, 1], 'r--') # Courbe au hasard*

*plt.xlabel('Taux de Faux Positifs')*

*plt.ylabel('Taux de Vrais Positifs')*

*plt.title('Courbe ROC')*

*plt.legend()*

*plt.show()*

## **4. Conclusion**

* **Matrice de confusion** : Base de l’évaluation des modèles de classification.
* **Accuracy** : Bien pour des classes équilibrées.
* **Précision** : Utile si **les faux positifs sont critiques**.
* **Rappel** : Utile si **les faux négatifs sont critiques**.
* **F1-score** : Bon compromis entre précision et rappel.
* **Courbe ROC & AUC** : Mesure la capacité du modèle à discriminer les classes.

1. **Clustering et Réduction de Dimension**

Le **clustering** et la **réduction de dimension** sont deux concepts clés en apprentissage non supervisé.

* **Clustering** : Regrouper des données similaires sans étiquettes prédéfinies.
* **Réduction de dimension** : Réduire le nombre de variables tout en conservant l’essentiel de l’information.

Le **clustering** permet d’identifier des structures cachées dans les données en regroupant des observations similaires.

* + Algorithmes de clustering : **K-Means**, **DBSCAN**.

Le **clustering** est une technique d’apprentissage non supervisé qui permet de **regrouper** des données en fonction de leur similarité. Deux algorithmes très utilisés sont :

* **K-Means** : Basé sur la minimisation de la variance intra-cluster.
* **DBSCAN** : Basé sur la densité des points.

## **1. K-Means (K-Moyennes)**

### ****1.1 Principe de fonctionnement****

L’algorithme **K-Means** divise les données en **k** groupes de manière à minimiser la variance intra-cluster. Il suit les étapes suivantes :

1. Choisir **k** centres de clusters aléatoirement.
2. Assigner chaque point au **centre le plus proche**.
3. Calculer un **nouveau centre de cluster** (moyenne des points).
4. Répéter jusqu’à convergence.

### ****1.2 Avantages et Inconvénients****

**Avantages**

* Simple et efficace sur des données bien séparées.
* Rapide sur des jeux de données de grande taille.

**Inconvénients**

* Doit choisir **k** à l’avance.
* Sensible aux **outliers** et aux **clusters non sphériques**.

### ****1.3 Exemple Python avec K-Means****

*import numpy as np*

*import matplotlib.pyplot as plt*

*from sklearn.cluster import KMeans*

*from sklearn.datasets import make\_blobs*

*# Générer des données simulées*

*X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.6, random\_state=42)*

*# Appliquer K-Means*

*kmeans = KMeans(n\_clusters=4, random\_state=42)*

*y\_kmeans = kmeans.fit\_predict(X)*

*# Visualiser les clusters*

*plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, cmap='viridis')*

*plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=200, c='red', marker='X')*

*plt.title("Clustering avec K-Means")*

*plt.show()*

## **2. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**

### ****2.1 Principe de fonctionnement****

DBSCAN est un algorithme de clustering basé sur la **densité des points** :

1. Définir un **rayon (eps)** et un **nombre minimum de points (min\_samples)**.
2. Trouver les **points denses** (core points).
3. Étendre les clusters à partir des core points.
4. Détecter les **outliers** (points isolés).

### ****2.2 Avantages et Inconvénients****

**Avantages**

* Détecte des **formes de clusters complexes**.
* Gère bien les **outliers**.
* Pas besoin de définir **k** clusters.

**Inconvénients**

* Sensible aux choix de **eps** et **min\_samples**.
* Moins efficace sur des **données à haute dimension**.

### ****2.3 Exemple Python avec DBSCAN****

*from sklearn.cluster import DBSCAN*

*# Appliquer DBSCAN*

*dbscan = DBSCAN(eps=0.5, min\_samples=5)*

*y\_dbscan = dbscan.fit\_predict(X)*

*# Visualiser les clusters*

*plt.scatter(X[:, 0], X[:, 1], c=y\_dbscan, cmap='plasma')*

*plt.title("Clustering avec DBSCAN")*

*plt.show()*

## **3. Comparaison K-Means vs DBSCAN**

| **Critère** | **K-Means** | **DBSCAN** |
| --- | --- | --- |
| Type de clustering | Partitionné | Basé sur la densité |
| Forme des clusters | Sphériques | Formes variées |
| Gestion des outliers | Mauvaise | Bonne |
| Nombre de clusters | Doit être fixé | Automatique |
| Sensibilité aux paramètres | Moyenne (choix de k) | Élevée (eps, min\_samples) |
| Rapidité | Rapide | Plus lent sur grandes données |

## **4. Conclusion**

* **K-Means** est rapide et efficace pour des **clusters bien séparés**.
* **DBSCAN** est plus flexible et détecte **les outliers**.
* Le choix de l’algorithme dépend de **la structure des données**.
  + PCA et techniques de réduction de dimension.

# **PCA et Techniques de Réduction de Dimension**

La **réduction de dimension** est une technique essentielle en Machine Learning et en Data Science. Elle permet de réduire le nombre de variables tout en conservant l’essentiel de l’information. Cela facilite la **visualisation des données**, accélère les **calculs** et peut **éviter le sur-apprentissage**.

## **1. Pourquoi réduire la dimensionnalité ?**

**Avantages** :

* Diminue la **complexité des modèles**.
* Évite le **problème de malédiction de la dimension**.
* Améliore la **visualisation** des données.
* Diminue le **bruit et les corrélations inutiles**.

**Inconvénients** :

* Perte possible d’**interprétabilité**.
* Risque de perdre de l’**information utile**.

## **2. PCA (Analyse en Composantes Principales)**

### ****2.1 Principe****

La **PCA (Principal Component Analysis)** est une méthode qui transforme des **variables corrélées** en **nouvelles variables indépendantes** appelées **composantes principales**.

* Elle fonctionne en trouvant les **axes de variance maximale** dans les données.
* Elle projette les données sur ces axes tout en minimisant la **perte d'information**.

**Étapes de la PCA** :

1. Centrer et normaliser les données.
2. Calculer la matrice de covariance.
3. Trouver les **valeurs propres** et **vecteurs propres**.
4. Sélectionner les **composantes principales**.
5. Projeter les données sur ces nouvelles dimensions.

### ****2.2 Exemple Python avec PCA****

*import numpy as np*

*import matplotlib.pyplot as plt*

*from sklearn.decomposition import PCA*

*from sklearn.datasets import load\_digits*

*import seaborn as sns*

*# Charger un jeu de données (exemple : chiffres manuscrits)*

*digits = load\_digits()*

*X = digits.data*

*# Réduction à 2 dimensions*

*pca = PCA(n\_components=2)*

*X\_pca = pca.fit\_transform(X)*

*# Visualisation des données projetées*

*plt.figure(figsize=(8, 6))*

*sns.scatterplot(x=X\_pca[:, 0], y=X\_pca[:, 1], hue=digits.target, palette="tab10")*

*plt.title("Réduction de Dimension avec PCA")*

*plt.show()*

### ****2.3 Choix du Nombre de Composantes****

On peut choisir le **nombre optimal de composantes** en regardant **l’explication de la variance** :

*# Calcul de la variance expliquée*

*explained\_variance = np.cumsum(pca.explained\_variance\_ratio\_)*

*plt.figure(figsize=(8, 5))*

*plt.plot(range(1, len(explained\_variance) + 1), explained\_variance, marker='o', linestyle='--')*

*plt.xlabel('Nombre de Composantes')*

*plt.ylabel('Variance Cumulative')*

*plt.title("Choix du Nombre de Composantes PCA")*

*plt.show()*

**Règle courante** : Choisir le nombre de composantes qui capture **au moins 90%** de la variance.

## **3. Autres Techniques de Réduction de Dimension**

### ****3.1 t-SNE (t-Distributed Stochastic Neighbor Embedding)****

* Idéal pour la **visualisation en 2D ou 3D**.
* Préserve les relations locales entre les points.
* **Limite** : Plus lent et non interprétable pour l’entraînement de modèles.

*from sklearn.manifold import TSNE*

*tsne = TSNE(n\_components=2, random\_state=42)*

*X\_tsne = tsne.fit\_transform(X)*

*plt.scatter(X\_tsne[:, 0], X\_tsne[:, 1], c=digits.target, cmap='tab10')*

*plt.title("Visualisation avec t-SNE")*

*plt.show()*

### ****3.2 UMAP (Uniform Manifold Approximation and Projection)****

* Plus rapide que t-SNE.
* Bonne généralisation pour la visualisation et la classification.

*import umap*

*umap\_reducer = umap.UMAP(n\_components=2)*

*X\_umap = umap\_reducer.fit\_transform(X)*

*plt.scatter(X\_umap[:, 0], X\_umap[:, 1], c=digits.target, cmap='tab10')*

*plt.title("Visualisation avec UMAP")*

*plt.show()*

## **4. Comparaison des Méthodes**

| **Méthode** | **Utilisation** | **Avantages** | **Inconvénients** |
| --- | --- | --- | --- |
| **PCA** | Réduction de dimension, prétraitement | Rapide, interprétable | Linéaire, perte d'info |
| **t-SNE** | Visualisation | Très bon pour les données complexes | Lent, pas généralisable |
| **UMAP** | Visualisation et classification | Rapide, efficace sur grands jeux de données | Paramètres sensibles |

## **5. Conclusion**

* **PCA** : Idéal pour la **réduction de dimension** et la **visualisation basique**.
* **t-SNE** et **UMAP** : Plus adaptés à la **visualisation avancée**.
* La bonne technique dépend des **besoins spécifiques du projet**.

1. **Optimisation et Validation des Modèles**

L'optimisation et la validation sont des étapes clés en **Machine Learning** pour s'assurer qu'un modèle est **précis**, **généralisable** et **performant** sur des données non vues.

## **1. Validation des Modèles**

L’objectif est d’évaluer la performance d’un modèle avant de le déployer.

### ****1.1 Séparation des Données : Train/Test Split****

On divise les données en **deux ensembles** :

* **Train (80%)** : Sert à entraîner le modèle.
* **Test (20%)** : Permet d’évaluer la performance du modèle.

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.datasets import load\_iris*

*# Charger un dataset*

*X, y = load\_iris(return\_X\_y=True)*

*# Séparer en train et test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

### ****1.2 Validation Croisée (Cross-Validation - CV)****

La **cross-validation** permet d’évaluer un modèle plus précisément en répétant l’apprentissage plusieurs fois avec des **sous-ensembles différents**.

**Méthode K-Fold** :

1. Diviser les données en **k** groupes (ex : k=5).
2. Entraîner le modèle sur **k-1** groupes.
3. Tester sur le groupe restant.
4. Répéter k fois et calculer la **moyenne des performances**.

*from sklearn.model\_selection import cross\_val\_score*

*from sklearn.ensemble import RandomForestClassifier*

*model = RandomForestClassifier()*

*scores = cross\_val\_score(model, X, y, cv=5) # 5-Fold Cross Validation*

*print("Scores :", scores)*

*print("Score moyen :", scores.mean())*

**Avantages** : Plus robuste, évite le sur-apprentissage.  
**Inconvénients** : Plus lent car entraîne plusieurs modèles.

## **2. Optimisation des Modèles**

L’optimisation permet d’améliorer la performance en **ajustant les hyperparamètres** du modèle.

### ****2.1 Recherche par Grille (Grid Search)****

Teste **toutes les combinaisons** possibles d’hyperparamètres.

*from sklearn.model\_selection import GridSearchCV*

*param\_grid = {'n\_estimators': [50, 100, 200], 'max\_depth': [3, 5, 10]}*

*grid\_search = GridSearchCV(RandomForestClassifier(), param\_grid, cv=5)*

*grid\_search.fit(X\_train, y\_train)*

*print("Meilleurs paramètres :", grid\_search.best\_params\_)*

### ****2.2 Recherche aléatoire (Randomized Search)****

Teste un **échantillon aléatoire** d’hyperparamètres, plus rapide que Grid Search.

*from sklearn.model\_selection import RandomizedSearchCV*

*from scipy.stats import randint*

*param\_dist = {'n\_estimators': randint(50, 200), 'max\_depth': randint(3, 10)}*

*random\_search = RandomizedSearchCV(RandomForestClassifier(), param\_dist, cv=5, n\_iter=10)*

*random\_search.fit(X\_train, y\_train)*

*print("Meilleurs paramètres :", random\_search.best\_params\_)*

## **3. Évaluation des Modèles**

### ****3.1 Métriques pour la Régression****

* **MSE (Mean Squared Error)** : Erreur quadratique moyenne.
* **RMSE (Root Mean Squared Error)** : Racine carrée de MSE.
* **R² Score** : Qualité du modèle à expliquer la variance.

*from sklearn.metrics import mean\_squared\_error, r2\_score*

*y\_pred = grid\_search.predict(X\_test)*

*print("MSE :", mean\_squared\_error(y\_test, y\_pred))*

*print("R² :", r2\_score(y\_test, y\_pred))*

### ****3.2 Métriques pour la Classification****

* **Accuracy** : % de bonnes prédictions.
* **Précision / Rappel** : Utiles pour données déséquilibrées.
* **Matrice de confusion** : Visualise les erreurs du modèle.

*from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix*

*y\_pred = grid\_search.predict(X\_test)*

*print("Accuracy :", accuracy\_score(y\_test, y\_pred))*

*print("Rapport de classification :\n", classification\_report(y\_test, y\_pred))*

*print("Matrice de confusion :\n", confusion\_matrix(y\_test, y\_pred))*

## **4. Conclusion**

* **Validation** : Utiliser **train/test split** et **cross-validation**.
* **Optimisation** : Tester les hyperparamètres avec **Grid Search** ou **Randomized Search**.
* **Évaluation** : Choisir les **bonnes métriques** selon le type de problème (**régression** ou **classification**).