Université Moulay Ismail
ENS, Meknès
Département des Sciences
A.U : 2024–2025

IA et AS
TD/TP N : 1
Filière : Master : SIE
Pr. ABDELLAOUI

TD/TP 1 : Machine Learning

Objectif

Ce TP vous guidera à travers un processus complet d'exploration et d'analyse de données, en commençant par les techniques fondamentales jusqu'aux méthodes avancées. Vous apprendrez à :

- Comprendre la structure et la qualité de vos données
- Visualiser et interpréter les distributions et relations
- Détecter et traiter les anomalies
- Préparer vos données pour la modélisation

Partie 1
Rappel de concepts de base

1 Bibliothèques Essentielles pour le Machine Learning

1.1 1. NumPy (Numerical Python)

Import : `import numpy as np`

- Bibliothèque fondamentale pour le calcul scientifique
- Gestion efficace des tableaux multidimensionnels (arrays)
- Opérations mathématiques vectorisées
- Fonctions mathématiques avancées
- Base pour la manipulation des données en ML

Exemples d'utilisation :

```
import numpy as np

# Création d'arrays
array = np.array([1, 2, 3, 4, 5])
matrix = np.zeros((3, 3))

# Opérations mathématiques
result = np.mean(array)
dot_product = np.dot(matrix, matrix)
```

1.2 Matplotlib

Import Matplotlib

Import : `import matplotlib.pyplot as plt`

— Bibliothèque de visualisation de données
— Création de graphiques, histogrammes, scatter plots
— Personnalisation complète des visualisations
— Intégration avec Jupyter notebooks

Exemples d'utilisation :

```
import matplotlib.pyplot as plt

# Création d'un simple graphique
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])
plt.title("Mon graphique")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

1.3 Scikit-learn (sklearn)

Import Scikit-learn (sklearn)

Import : `from sklearn import [module]`

— Bibliothèque principale pour le machine learning
— Implémentation de nombreux algorithmes
— Outils de prétraitement des données
— Métriques d'évaluation
— Validation croisée

Exemples d'utilisation :

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Préparation des données
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Prétraitement
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Modélisation
model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

1.4 Pandas

Import Pandas

Import : `import pandas as pd`

— Manipulation et analyse de données structurées
— Gestion des DataFrames et Series
— Lecture/écriture de différents formats de données
— Fonctions d'agrégation et de transformation

Exemples d'utilisation :

```
import pandas as pd

# Lecture de données
df = pd.read_csv('data.csv')

# Manipulation de données
df_filtered = df[df['colonne'] > 5]
df_grouped = df.groupby('categorie').mean()
```

1.5 TensorFlow

Import : `import tensorflow as tf`

— Bibliothèque pour le deep learning
— Construction et entraînement de réseaux de neurones
— Calcul distribué
— Optimisé pour le GPU

Usage typique :

```
import tensorflow as tf

# Création d'un modèle simple
model = tf.keras.Sequential([
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(10, activation='softmax')
])
```

1.6 Keras

Import : `from tensorflow import keras`

— Interface haut niveau pour les réseaux de neurones
— API intuitive pour le deep learning
— Intégré à TensorFlow
— Rapide à prototyper

Exemple d'utilisation :

```
    from tensorflow import keras

    model = keras.Sequential([
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
```

1.7 Points Importants

— Ces bibliothèques sont complémentaires et souvent utilisées ensemble
— NumPy et Pandas pour la préparation des données
— Matplotlib pour la visualisation
— Scikit-learn pour le ML classique
— TensorFlow/Keras pour le deep learning
— Installation via Anaconda recommandée pour la compatibilité

2 Fonctions Essentielles Pandas pour l'Analyse de Données

2.1 Fonctions d'Exploration des Données

```
    # 1. Visualisation des données
    df.head(n)      # Affiche les n premières lignes (par défaut 5)
    df.tail(n)      # Affiche les n dernières lignes
    df.sample(n)    # Affiche n lignes aléatoires

    # 2. Informations sur le DataFrame
    df.info()       # Affiche les informations générales (types, mémoire,
       etc.)
    df.shape        # Retourne un tuple (nombre de lignes, nombre de colonnes)
    df.columns      # Liste des noms de colonnes
    df.index        # Liste des index
    df.dtypes       # Types de données de chaque colonne

    # 3. Statistiques descriptives
    df.describe()  # Statistiques de base pour les colonnes numériques
    df.describe(include=['object'])  # Statistiques pour les colonnes
       catégorielles
    df.count()      # Nombre de valeurs non-NULL par colonne
    df.mean()       # Moyenne de chaque colonne numérique
    df.median()     # Médiane
    df.std()        # Écart-type
    df.min()        # Minimum
    df.max()        # Maximum
    df.quantile([0.25, 0.75])  # Quartiles spécifiques
```

2.2 Nettoyage et Manipulation des Données

```
# 1. Gestion des valeurs manquantes
df.isnull()    # Retourne un DataFrame de booléens (True pour NaN)
df.isnull().sum()  # Compte les valeurs manquantes par colonne
df.dropna()    # Supprime les lignes avec valeurs manquantes
df.fillna(value)   # Remplace les valeurs manquantes par 'value'

# 2. Filtrage des données
df[df['colonne'] > valeur]  # Filtre les lignes selon une condition
df.query('colonne > valeur')  # Méthode alternative de filtrage
df.loc[condition]  # Sélection par label avec condition
df.iloc[indices]   # Sélection par position

# 3. Modification des colonnes
df.rename(columns={'ancien': 'nouveau'})  # Renommer des colonnes
df.drop(['colonne'], axis=1)  # Supprimer des colonnes
df.drop_duplicates()  # Supprimer les doublons
```

2.3 Agrégation et Groupement

```
# 1. Groupement
df.groupby('colonne').mean()  # Moyenne par groupe
df.groupby(['col1', 'col2']).sum()  # Groupement multiple
df.groupby('colonne').agg(['mean', 'count'])  # Multiple agrégations

# 2. Pivots et Reshaping
df.pivot_table(values='val', index='idx', columns='cols')
df.melt(id_vars=['id'], value_vars=['val1', 'val2'])

# 3. Tri
df.sort_values('colonne')  # Tri par valeurs
df.sort_values(['col1', 'col2'], ascending=[True, False])
```

2.4 Fusion et Combinaison de Données

```
# 1. Fusion de DataFrames
pd.concat([df1, df2])  # Concaténation verticale
pd.concat([df1, df2], axis=1)  # Concaténation horizontale
df1.merge(df2, on='key')  # Fusion SQL-style
df1.join(df2)  # Fusion basée sur l'index

# 2. Opérations sur les colonnes
df['nouvelle'] = df['col1'] + df['col2']  # Calculs entre colonnes
df.assign(nouvelle=lambda x: x['col1']*2)  # Ajout de colonne
```

2.5 Analyses Statistiques et Calculs

```
    # 1. Corrélations
    df.corr()  # Matrice de corrélation
    df['col1'].corr(df['col2'])  # Corrélation entre deux colonnes

    # 2. Fonctions de fenêtrage
    df['col'].rolling(window=3).mean()  # Moyenne mobile
    df['col'].expanding().sum()  # Somme cumulée
    df['col'].shift(1)  # Décalage des valeurs

    # 3. Résumés statistiques avancés
    df.value_counts()  # Comptage des valeurs uniques
    df.nunique()  # Nombre de valeurs uniques
    df.mode()  # Mode statistique
```

2.6 Export et Import de Données

```
    # 1. Lecture de données
    pd.read_csv('fichier.csv')
    pd.read_excel('fichier.xlsx')
    pd.read_sql('query', connection)

    # 2. Export de données
    df.to_csv('fichier.csv')
    df.to_excel('fichier.xlsx')
    df.to_sql('table', connection)
```

2.7 Exemples Pratiques

```
    # Exemple complet d'analyse
    import pandas as pd

    # Chargement et exploration
    df = pd.read_csv('donnees.csv')
    print("Aperçu des données:")
    print(df.head())
    print("\nInformations sur le dataset:")
    print(df.info())

    # Statistiques descriptives
    print("\nStatistiques descriptives:")
    print(df.describe())

    # Vérification des valeurs manquantes
    print("\nValeurs manquantes par colonne:")
    print(df.isnull().sum())

    # Groupement et agrégation
    resume = df.groupby('categorie').agg({
        'valeur': ['mean', 'std', 'count'],
        'prix': ['sum', 'mean']
```

```
    })
    print("\nRésumé par catégorie:")
    print(resume)

    # Corrélations
    print("\nMatrice de corrélation:")
    print(df.corr())
```

Partie 2

TP : Modélisation Prédictive des Prix de Maisons

Objectif

Dans cette partie, vous allez appliquer les connaissances acquises en analyse exploratoire pour construire un modèle prédictif des prix de maisons en Californie. Vous suivrez toutes les étapes d'un projet de machine learning, de la préparation des données jusqu'au déploiement du modèle.

TAF 1:

Chargement des Données

```
# -*- coding: utf-8 -*-
from sklearn.datasets import fetch_california_housing
import pandas as pd
import numpy as np

# Chargement des données
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['MedHouseVal'] = data.target

# Affichage des premières lignes

print("Aperçu des données:")
print(df.head())
print("\nInformations sur le dataset:")
print(df.info())
```

3 Exercices d'Exploration des Données

OBJECTIFS

1. Exploration Basique des donnees
2. Analyse des Relations
3. Analyse des Distributions
4. Filtrage et Agrégation
5. Détection d'Anomalies
6. Création de Features et Analyse Approfondie

3.1 Niveau Débutant

1. **Exploration Basique**
   (a) Combien de lignes et de colonnes contient le dataset ?
   (b) Quels sont les noms de toutes les colonnes ?
   (c) Quels sont les types de données de chaque colonne ?
   (d) Y a-t-il des valeurs manquantes dans le dataset ?
2. **Statistiques Descriptives**
   (a) Quelle est la moyenne du prix des maisons ?
   (b) Quel est l'âge moyen des maisons ?
   (c) Quelles sont les valeurs minimales et maximales pour chaque feature ?
   (d) Calculez l'écart-type de chaque variable numérique
3. **Visualisation Simple**
   (a) Créez un histogramme de la distribution des prix
   (b) Affichez un boxplot pour chaque variable numérique
   (c) Visualisez la répartition géographique (latitude/longitude) des maisons

3.2 Niveau Intermédiaire

1. **Analyse des Relations**
   (a) Calculez la corrélation entre toutes les variables numériques
   (b) Identifiez les 3 features les plus corrélées avec le prix
   (c) Créez des scatter plots entre le prix et ces features
2. **Analyse des Distributions**
   (a) Évaluez la normalité de chaque variable numérique
   (b) Calculez le skewness (asymétrie) de chaque variable
   (c) Identifiez les variables qui nécessiteraient une transformation
3. **Filtrage et Agrégation**
   (a) Filtrez les maisons dont le prix est supérieur à la moyenne
   (b) Groupez les données par région et calculez le prix moyen
   (c) Identifiez les zones les plus chères et les moins chères

3.3 Niveau Avancé

1. **Détection d'Anomalies**
   (a) Utilisez la méthode IQR pour détecter les outliers
   (b) Calculez le Z-score pour chaque variable
   (c) Proposez une stratégie de traitement des outliers
2. **Création de Features**
   (a) Créez un ratio prix/surface
   (b) Discretisez les variables continues en catégories
   (c) Proposez de nouvelles features pertinentes
3. **Analyse Approfondie**
   (a) Évaluez l'impact de la localisation sur les prix

(b) Analysez les interactions entre variables

(c) Réalisez une analyse en composantes principales (PCA)

TAF 2:

Préparation des Données

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Séparation features/cible
X = df.drop('MedHouseVal', axis=1)
y = df['MedHouseVal']

# Normalisation
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Division train/test
X_train, X_test, y_train, y_test = train_test_split(
X_scaled, y, test_size=0.2, random_state=42)
```

4 Exercices de Préparation des Données

OBJECTIFS

1. Questions de base sur la séparation features/cible, normalisation et division train/test
2. StandardScaler : fonctionnement, importance et application
3. Ordre optimal des opérations de prétraitement
4. Gestion des valeurs aberrantes et validation des données
5. Choix et impact des différents types de scalers
6. Considérations pour le déploiement en production

4.1 Questions de Base

1. **Compréhension Générale**
   (a) Pourquoi sépare-t-on les données en features (X) et cible (y) ?
   (b) Que signifie `axis=1` dans `df.drop('MedHouseVal', axis=1)` ?
   (c) Quelle est la différence entre X et X_scaled ?
   (d) Pourquoi utilisons-nous `random_state=42` ?
2. **Division Train/Test**
   (a) Que signifie `test_size=0.2` ?
   (b) Calculez le nombre d'échantillons dans les ensembles train et test
   (c) Pourquoi est-il important de diviser les données en train et test ?
   (d) Dans quel ordre doit-on faire la normalisation et la division train/test ?
3. **Normalisation**
   (a) Qu'est-ce que le StandardScaler ?

(b) Comment fonctionne la normalisation ?
(c) Pourquoi normaliser les données ?
(d) Quelles variables doivent être normalisées ?

4.2 Questions Intermédiaires

1. **StandardScaler en Détail**
   (a) Quelle est la différence entre `fit`, `transform` et `fit_transform` ?
   (b) Comment le StandardScaler calcule-t-il la normalisation ?
   (c) Quelle est la moyenne et l'écart-type des données après normalisation ?
   (d) Comment peut-on inverser la normalisation ?
2. **Validation des Données**
   (a) Comment vérifier que la normalisation a fonctionné correctement ?
   (b) Comment s'assurer que la division train/test est représentative ?
   (c) Quelle est l'importance de la stratification dans le split ?
   (d) Comment traiter les outliers avant ou après la normalisation ?
3. **Alternatives et Choix**
   (a) Quelles sont les alternatives au StandardScaler ?
   (b) Dans quels cas utiliser MinMaxScaler plutôt que StandardScaler ?
   (c) Quels sont les avantages et inconvénients d'un test_size de 0.2 ?
   (d) Comment choisir la taille de l'ensemble de test ?

4.3 Questions Avancées

1. **Impact sur le Modèle**
   (a) Comment la normalisation affecte-t-elle différents types de modèles ?
   (b) Quel est l'impact du random_state sur les résultats ?
   (c) Comment gérer la normalisation pour de nouvelles données ?
   (d) Comment la normalisation influence-t-elle les performances du modèle ?
2. **Validation Croisée**
   (a) Comment intégrer la validation croisée dans ce processus ?
   (b) Où placer la normalisation dans un pipeline avec validation croisée ?
   (c) Comment éviter le data leakage lors de la normalisation ?
   (d) Quelles sont les meilleures pratiques pour la validation des données ?
3. **Optimisation et Production**
   (a) Comment sauvegarder et charger un scaler ?
   (b) Comment intégrer la préparation des données dans un pipeline ?
   (c) Comment gérer les mises à jour du modèle en production ?
   (d) Quelles sont les considérations de performance pour le scaling en production ?

4.4 Exercices Pratiques

1. **Exercice 1 : Comparaison des Scalers**

```
        # Comparez différents scalers
        from sklearn.preprocessing import MinMaxScaler, RobustScaler
        # Implémentez la comparaison
```

2. **Exercice 2 : Validation de la Normalisation**

```
        # Vérifiez les propriétés des données normalisées
        print("Moyenne:", X_scaled.mean(axis=0))
        print("Écart-type:", X_scaled.std(axis=0))
```

3. **Exercice 3 : Pipeline de Préparation**

```
        from sklearn.pipeline import Pipeline
        # Créez un pipeline complet de préparation des données
```

4.5 Points de Discussion

— Discutez de l'importance de la reproductibilité dans la préparation des données

— Analysez les compromis entre différentes approches de scaling

— Évaluez l'impact des choix de préparation sur les résultats du modèle

— Proposez des améliorations possibles au processus de préparation

Points d'Attention

Points importants pour la préparation des données :

— La normalisation est cruciale pour de nombreux algorithmes

— La division train/test permet d'évaluer la généralisation

— La gestion des valeurs aberrantes peut impacter les performances

TAF 3:

Entraînement du Modèle

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Entraînement
model = LinearRegression()
model.fit(X_train, y_train)

# Prédictions
y_pred = model.predict(X_test)

# Évaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

5 Exercices d'Entraînement du Modèle

OBJECTIFS

1. Compréhension de la régression linéaire et ses méthodes
2. Métriques d'évaluation (MSE, R, MAE)
3. Analyse des coefficients et leur interprétation
4. Diagnostic du modèle et visualisation des résultats
5. Analyse des résidus et validation
6. Techniques avancées :
   - Validation croisée
   - Analyse de sensibilité
   - Intervalles de confiance par bootstrap
   - Gestion du compromis biais-variance
   - Stratégies d'optimisation du modèle

5.1 Questions Fondamentales

1. **Compréhension du Modèle**
   (a) Qu'est-ce qu'une régression linéaire ?
   (b) Comment fonctionne la méthode `fit()` ?
   (c) Que fait la méthode `predict()` ?
   (d) Pourquoi utilise-t-on X_train et y_train pour l'entraînement ?
2. **Métriques d'Évaluation**
   (a) Que mesure le Mean Squared Error (MSE) ?
   (b) Que représente le score R^2 ?
   (c) Quelles sont les valeurs optimales pour ces métriques ?
   (d) Comment interpréter ces scores dans le contexte de la prédiction des prix de maisons ?

3. **Exercice Pratique de Base**

```
        # Calculez l'erreur moyenne absolue (MAE)
        from sklearn.metrics import mean_absolute_error
        mae = mean_absolute_error(y_test, y_pred)
        print(f"Mean Absolute Error: {mae}")
```

5.2 Questions Intermédiaires

1. **Analyse des Coefficients**
   (a) Comment accéder aux coefficients du modèle ?
   (b) Comment interpréter les coefficients de la régression ?
   (c) Quelle feature a le plus d'impact sur les prédictions ?
   (d) Comment calculer l'importance relative des features ?
2. **Diagnostic du Modèle**

```
        # Exemple de code pour l'analyse des coefficients
        coefficients = pd.DataFrame({
            'Feature': X.columns,
            'Coefficient': model.coef_
        })
        print("Coefficients du modèle:")
        print(coefficients.sort_values(by='Coefficient',
        ascending=False))
```

3. **Visualisation des Résultats**

```
        import matplotlib.pyplot as plt

        plt.scatter(y_test, y_pred)
        plt.plot([y_test.min(), y_test.max()],
        [y_test.min(), y_test.max()], 'r--')
        plt.xlabel('Prix Réels')
        plt.ylabel('Prix Prédits')
        plt.title('Comparaison Prédictions vs Réalité')
        plt.show()
```

5.3 Questions Avancées

1. **Analyse des Résidus**
   — Comment calculer et analyser les résidus ?
   — Que nous apprend la distribution des résidus ?
   — Comment détecter les biais systématiques dans les prédictions ?
   — Comment utiliser les résidus pour améliorer le modèle ?
2. **Validation du Modèle**

```
        # Analyse des résidus
        residus = y_test - y_pred
        plt.hist(residus, bins=30)
        plt.title('Distribution des Résidus')
        plt.xlabel('Erreur de Prédiction')
```

```
        plt.ylabel('Fréquence')
        plt.show()

        # Test de normalité des résidus
        from scipy import stats
        _, p_value = stats.normaltest(residus)
        print(f"P-value du test de normalité: {p_value}")
```

3. **Questions d'Optimisation**
   - Comment pourrait-on améliorer les performances du modèle ?
   - Quels sont les avantages et inconvénients de la régression linéaire ?
   - Quels autres modèles pourraient être plus appropriés ?
   - Comment gérer le compromis biais-variance ?

5.3.1 Exercices Pratiques Avancés

1. **Validation Croisée**

```
        from sklearn.model_selection import cross_val_score

        # Calculer les scores de validation croisée
        cv_scores = cross_val_score(model, X_scaled, y,
        cv=5, scoring='r2')
        print(f"Scores CV: {cv_scores}")
        print(f"Score moyen: {cv_scores.mean()}")
        print(f"Écart-type: {cv_scores.std()}")
```

2. **Analyse de Sensibilité**

```
        # Analyse de l'impact de chaque feature
        for i, feature in enumerate(X.columns):
        impact = abs(model.coef_[i] * X_scaled[:, i].std())
        print(f"Impact de {feature}: {impact}")
```

3. **Intervalles de Confiance**

```
        from sklearn.utils import resample

        # Bootstrap pour estimer l'incertitude
        n_iterations = 1000
        scores = []
        for i in range(n_iterations):
        # Rééchantillonnage
        X_boot, y_boot = resample(X_scaled, y)
        # Ajustement et score
        score = model.fit(X_boot, y_boot).score(X_scaled, y)
        scores.append(score)

        print(f"Intervalle de confiance R squar : "
        f"[{np.percentile(scores, 2.5):.3f}, "
        f"{np.percentile(scores, 97.5):.3f}]")
```

5.3.2 Questions de Réflexion

— Comment ce modèle se comporterait-il avec des données très différentes ?
— Quels sont les signes d'un surapprentissage ou sous-apprentissage ?
— Comment pourriez-vous expliquer ce modèle à un non-spécialiste ?
— Quelles hypothèses de la régression linéaire sont importantes à vérifier ?

TAF 4:

Mini Projets