Compte rendu

JAMAL YASSINE
Numéro d'étudiant : 22007655
Classe : CAC2

GRAND GUIDE : ANATOMIE D'UN PROJET DATA SCIENCE - DIGITS

Ce document décortique chaque étape du cycle de vie d'un projet de Machine Learning appliqué au dataset Digits. Il a pour objectif de faire évoluer le lecteur d'un profil de "débutant qui exécute du code" vers celui d'un "praticien qui conçoit, comprend et justifie chaque choix technique du pipeline".

---

1 Le Contexte Métier et la Mission

1.1 Le Problème (Business Case)

Dans le domaine de la reconnaissance optique de caractères (OCR), identifier automatiquement des chiffres manuscrits accélère le traitement de documents scannés (factures, formulaires bancaires).

- **Objectif :** Créer un "Assistant IA" pour lire automatiquement les chiffres manuscrits 0-9.
- **L'Enjeu critique :** La matrice des coûts d'erreur est asymétrique.
    - Dire "1" au lieu de "7" = erreur bancaire.
    - Dire "0" au lieu de "6" = erreur de lecture de compte.
    - **L'IA doit prioriser la précision globale (>95%).**

1.2 Les Données (L'Input)

Dataset **Digits** de Scikit-Learn.

- **X (Features) :** 64 colonnes (pixels d'images 8x8 aplaties). Intensités 0-16.
- **y (Target) :** Multi-classe 0-9 (10 chiffres manuscrits).
- **Taille :** 1797 images.

---

2 Le Code Python (Laboratoire)

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Configuration
sns.set_theme(style="whitegrid")
import warnings
warnings.filterwarnings('ignore')

# --- PHASE 1 : ACQUISITION & SIMULATION ---
data = load_digits()
df = pd.DataFrame(data.data, columns=[f"pixel_{i}" for i in range(data.data.shape[1])])
df['target'] = data.target

# Simulation de la realite (Donnees sales) - 5% NaN
np.random.seed(42)
df_dirty = df.copy()
for col in df.columns[:-1]:
    df_dirty.loc[df_dirty.sample(frac=0.05).index, col] = np.nan

# --- PHASE 2 : DATA WRANGLING (NETTOYAGE) ---
X = df_dirty.drop('target', axis=1)
y = df_dirty['target']

# Strategie d'imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
X_clean = pd.DataFrame(X_imputed, columns=X.columns)

# --- PHASE 3 : ANALYSE EXPLORATOIRE (EDA) ---
print("--- Statistiques Descriptives ---")
print(X_clean.iloc[:, :10].describe())

# Visualisation images
plt.figure(figsize=(10, 3))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(data.images[i], cmap="gray")
    plt.title(f"Label : {data.target[i]}")
    plt.axis("off")
plt.suptitle("Exemples d'images Digits", fontsize=14)
plt.tight_layout()
plt.show()

# --- PHASE 4 : PROTOCOLE EXPERIMENTAL (SPLIT) ---
X_train, X_test, y_train, y_test = train_test_split(
    X_clean, y, test_size=0.2, random_state=42, stratify=y
)

# --- PHASE 5 : INTELLIGENCE ARTIFICIELLE (RANDOM FOREST) ---
model = RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1)
model.fit(X_train, y_train)

# --- PHASE 6 : AUDIT DE PERFORMANCE ---
y_pred = model.predict(X_test)
```

```
62
63 print(f"\n--- Accuracy Globale : {accuracy_score(y_test, y_pred)*100:.2f}% ---")
64 print("\n--- Rapport Detaille ---")
65 print(classification_report(y_test, y_pred, target_names=[str(i) for i in range(10)]))
66
67 # Visualisation des erreurs
68 plt.figure(figsize=(8, 6))
69 sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
70 plt.title('Matrice de Confusion : Realite vs IA')
71 plt.ylabel('Vrai Chiffre')
72 plt.xlabel('Chiffre Predit')
73 plt.show()
```

---

3 Analyse Approfondie : Nettoyage (Data Wrangling)

3.1 Le Problème Mathématique du « Vide »

Les algorithmes de Machine Learning ne savent pas manipuler les valeurs `NaN`. Une seule valeur manquante dans un vecteur peut faire échouer le calcul de distances ou de probabilités. Dans ce projet, 5 % des valeurs de chaque feature ont été remplacées par `NaN`, soit plusieurs milliers de « trous » à combler avant d'entraîner un modèle.

3.2 La Mécanique de l'Imputation

On utilise `SimpleImputer(strategy="mean")` qui suit deux étapes :

1. **Apprentissage (`fit`) :** Pour chaque colonne `pixel_i`, l'algorithme calcule la moyenne μ_i des valeurs non manquantes et la stocke.
2. **Transformation (`transform`) :** Chaque `NaN` de `pixel_i` est remplacé par μ_i, ce qui donne une matrice `X_clean` entièrement numérique, sans valeurs manquantes.

Résultat : les modèles basés sur l'algèbre linéaire (et ici le Random Forest) peuvent fonctionner sans erreur liée aux `NaN`.

Le Coin de l'Expert (Data Leakage)

Dans ce script, le nettoyage est fait **avant** la séparation Train/Test.

- **Problème :** Les moyennes utilisées pour imputer le Test Set ont été calculées en utilisant également les données du Test $\rightarrow$ fuite d'information (« Data Leakage »).
- **Bonne pratique absolue :**
  - Faire le split Train/Test en premier.
  - Apprendre les moyennes (`fit`) uniquement sur le Train.
  - Appliquer la transformation (`transform`) sur le Test avec ces mêmes moyennes (souvent via un `Pipeline` Scikit-Learn).

4 Analyse Approfondie : Exploration (EDA)

4.1 Décrypter `.describe()`

Les statistiques descriptives sur les 10 premiers pixels donnent un premier profil du dataset Digits :

- Des pixels avec `mean = 0` et `std = 0` (comme certains pixels de bord) n'apportent aucune information utile, car ils sont toujours noirs.
- Des pixels centraux ont des moyennes et des écarts-types plus élevés, montrant qu'ils captent la forme des chiffres.

Comparer **mean** et **50 % (médiane)** permet de repérer des distributions asymétriques (skewness) : une moyenne beaucoup plus élevée que la médiane peut indiquer la présence de quelques intensités très fortes.

4.2 Les Visualisations Clés

Trois visualisations structurent le « profilage » :

- **Panel d'images 8×8** : montrer visuellement quelques chiffres (0–9) permet de relier les intensités de pixels à des formes concrètes.
- **Distribution de `pixel_20` par classe** : visualiser l'histogramme de ce pixel pour chaque chiffre permet de voir si ce pixel est discriminant.
- **Matrice de corrélation (20 premiers pixels)** : la heatmap met en évidence des groupes de pixels très corrélés, souvent voisins dans l'image, ce qui reflète la structure géométrique des chiffres manuscrits.

5 Analyse Approfondie : Méthodologie (Split)

5.1 Le Concept : La Garantie de Généralisation

Le but du Machine Learning est de **généraliser** sur de nouveaux chiffres manuscrits, pas d'apprendre par cœur les 1 797 exemples. Un split **80 % / 20 %** permet :

- D'avoir suffisamment de données pour apprendre la variabilité de l'écriture.
- De réserver un jeu de test indépendant pour estimer la performance en situation réelle.

5.2 Les Paramètres sous le Capot

L'appel à `train_test_split` utilise :

- `test_size=0.2` : $\approx$ 360 images de test.
- `random_state=42` : graine fixée pour un partitionnement reproductible.
- `stratify=y` : garantie que chaque classe (0–9) est représentée de manière équilibrée dans Train et Test.

Sans `stratify`, certaines classes rares pourraient être sous-représentées dans le jeu de test, faussant l'analyse des performances par chiffre.

6 FOCUS THÉORIQUE : L'Algorithme Random Forest

6.1 A. La Faiblesse de l'Arbre Isolé

Un seul arbre de décision apprend une succession de règles du type : « si tel pixel > seuil et tel autre pixel < seuil, alors chiffre = 3 ». Ce type de modèle a une **variance élevée** : il peut surapprendre des détails spécifiques à l'échantillon d'entraînement (overfitting).

6.2 B. La Force du Groupe (Bagging + Aléa)

Le Random Forest construit une forêt d'arbres hétérogènes grâce à deux sources d'aléa :

1. **Bootstrapping des données**
   Chaque arbre est entraîné sur un échantillon tiré avec remise du jeu d'entraînement (certains exemples sont répétés, d'autres absents).

2. **Aléa sur les features**
   À chaque split, l'arbre ne choisit la meilleure coupure qu'au sein d'un sous-ensemble aléatoire de pixels, ce qui diversifie les règles apprises.

En prédiction, les arbres votent, et la classe finale est choisie par **majorité**. Les erreurs individuelles se compensent, et le modèle final est plus stable.

6.3 C. Pourquoi Random Forest est adapté à Digits

- Il gère bien les **features corrélées** (pixels voisins dans l'image).
- Il supporte nativement les problèmes **multi-classes** (10 chiffres).
- Il est robuste au **bruit** et aux petites variations d'écriture.

7 Analyse Approfondie : Évaluation (L'Heure de Vérité)

7.1 A. Accuracy Globale

Le modèle Random Forest obtient une **accuracy supérieure à 95 %** sur le jeu de test, ce qui signifie que la majorité écrasante des chiffres manuscrits est correctement reconnue.

7.2 B. Rapport de Classification

Le rapport de classification (`classification_report`) donne, pour chaque chiffre de 0 à 9 :

- **Precision** : quand le modèle prédit ce chiffre, à quel point il a raison.
- **Recall** : parmi tous les exemples de ce chiffre, combien il en détecte correctement.
- **F1-score** : synthèse équilibrée des deux précédents.

Les scores sont élevés et relativement homogènes entre les classes, ce qui montre que le modèle ne se contente pas d'être bon sur une seule classe comme le 0 ou le 1, mais fonctionne bien sur l'ensemble des chiffres.

7.3 C. La Matrice de Confusion

La matrice de confusion (10×10) est visualisée sous forme de heatmap :

- La **diagonale** regroupe les prédictions correctes (vrai chiffre = chiffre prédit).
- Les **valeurs hors diagonale** révèlent les confusions (par exemple certains 3 pris pour des 5, certains 4 pris pour des 9).

Ces motifs d'erreurs donnent des pistes d'amélioration :

- Modèles plus spécialisés (par exemple CNN).
- Features additionnelles plus adaptées à la structure 2D des images.

Conclusion du Projet

Ce rapport montre que la Data Science ne s'arrête pas à `model.fit()`. C'est une chaîne de décisions logiques où :

- Le **contexte métier** (OCR) guide le choix des données (Digits 8×8) et des métriques (accuracy, F1 par chiffre).
- Le **pipeline technique** (simulation de NaN, imputation, EDA, split stratifié) prépare un terrain propre pour le modèle.
- L'algorithme **Random Forest** fournit une solution robuste et performante à un problème multi-classes réel.
- L'**audit de performance** (rapport de classification, matrice de confusion) permet d'interpréter les résultats et d'identifier les axes d'amélioration.