Projet de Data Science
Classification d'une maladie à partir de paramètres vitaux

Étudiante : SALIMA BLALET Groupe:CAC1 Module : Data Science / Machine Learning

1 Contexte métier et objectif

Dans le domaine médical, l'évaluation de l'état d'un patient repose souvent sur l'analyse de signaux vitaux tels que la température, le pouls, la saturation en oxygène, la glycémie et la tension artérielle. Ces signaux doivent être interprétés rapidement et de manière fiable, ce qui peut être complexe lorsque le volume de patients augmente ou que les symptômes sont peu spécifiques. [file :3][file :4][file :5]

L'objectif de ce projet est de construire un modèle de classification supervisée capable de prédire automatiquement si un patient est *sain* ou *malade* à partir de ses paramètres physiologiques. Dans ce contexte, les erreurs de type faux négatifs (patient malade prédit sain) sont particulièrement critiques, ce qui conduit à accorder une attention particulière au *rappel* (recall) de la classe malade, en plus de l'accuracy globale. [file :3][file :4][file :5]

2 Description du jeu de données

Le jeu de données utilisé, intitulé « maladie_observations », provient d'une source externe et est chargé au format CSV dans l'environnement Python. [file :4][file :5] Le dataset comporte 5 725 observations et 6 variables numériques : cinq variables explicatives (signaux vitaux) et une variable cible binaire. [file :4][file :5]

Les variables sont les suivantes. [file :4][file :5]

— **temperature** : température corporelle (en degrés Celsius, variable continue).
— **pouls** : fréquence cardiaque (battements par minute, variable continue).
— **oxygene** : saturation en oxygène du sang (en pourcentage, variable continue).
— **glycemie** : taux de glucose sanguin (variable continue).
— **tension** : tension artérielle, exprimée comme une valeur unique (variable continue).
— **label** : étiquette d'état de santé (0 = sain, 1 = malade).

La répartition de la cible montre un léger déséquilibre : 2 551 patients sains (`label = 0`) et 3 174 patients malades (`label = 1`). [file :4]

Code Python : chargement des données

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
plt.style.use('ggplot')
sns.set_theme(style="whitegrid")

# Chargement du dataset
data = pd.read_csv('/content/maladie_observations.csv')

# Aperçu et dimensions
print(data.sample(5))
print(data.shape)
print(data.groupby('label')['label'].value_counts())
```

[file :4][file :5]

3 Analyse exploratoire des données (EDA)

3.1 Qualité des données et valeurs manquantes

Une première analyse porte sur la présence de valeurs manquantes et sur les statistiques descriptives des variables. [file :4][file :5]

```
# Valeurs manquantes
print(data.isna().sum())

# Statistiques descriptives globales
print(data.describe())

# Focus sur la température
print(data.temperature.describe())
```

[file :4]

Cette étape met en évidence des valeurs manquantes dans `temperature`, `pouls` et `oxygene`, ainsi qu'un maximum aberrant de température autour de 522,5 °C, clairement incompatible avec des valeurs physiologiques. [file :4][file :5]

3.2 Correction des valeurs aberrantes

Afin de corriger l'outlier de température, les valeurs supérieures à un seuil logique (par exemple 100 °C) sont remplacées par la médiane de la température, calculée sur les valeurs plausibles. [file :4][file :5]

```
# Calcul de la médiane sur les valeurs plausibles
median_temp = data[data['temperature'] < 100]['temperature'].median()

# Remplacement des valeurs aberrantes
data['temperature'] = data['temperature'].apply(
    lambda x: median_temp if x > 100 else x
)

# Vérification après correction
print(data.temperature.describe())
```

[file :4]

Après cette correction, le maximum de température revient vers 40 °C, et la moyenne ainsi que l'écart-type redeviennent cohérents avec un contexte clinique réel. [file :4][file :5]

3.3 Imputation exploratoire de la température

À titre exploratoire, une imputation par la moyenne est également illustrée sur la colonne `temperature`. [file :4]

```
mean_temp = data['temperature'].mean()
data['temperature'] = data['temperature'].fillna(mean_temp)
print(data.temperature.describe())
```

[file :4]

3.4 Corrélations entre variables

Une matrice de corrélation est calculée et visualisée pour examiner les relations linéaires entre les signaux vitaux. [file :4]

```
plt.figure(figsize=(8, 6))
sns.heatmap(data.corr().abs(), cmap='magma', annot=True)
plt.title('Matrice de corrélation (valeurs absolues)')
plt.show()
```

[file :4]

Les corrélations observées restent modérées, et aucune colinéarité extrême ne remet en cause l'utilisation conjointe des cinq variables physiologiques dans le modèle. [file :4][file :5]

4 Nettoyage et préparation des données

4.1 Séparation des variables explicatives et de la cible

Les variables explicatives et la cible sont séparées comme suit. [file :4][file :5]

```
X = data.drop('label', axis=1)
Y = data['label']
```

[file :4]

4.2 Découpage en ensembles d'entraînement et de test

Les données sont scindées en un ensemble d'entraînement et un ensemble de test à l'aide d'un split 80/20. [file :4]

```
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=75
)
```

[file :4]

Le ratio 80/20 permet de disposer d'un volume suffisant pour l'apprentissage, tout en conservant un échantillon de test représentatif pour l'évaluation. Le paramètre `random_state` garantit la reproductibilité des résultats. [file :3][file :4]

4.3 Imputation des valeurs manquantes (bonne pratique)

Une imputation par la moyenne est ensuite appliquée sur les colonnes contenant des valeurs manquantes (`temperature`, `pouls`, `oxygene`). Pour éviter toute fuite de données (*data leakage*), l'imputer est ajusté uniquement sur l'ensemble d'entraînement, puis appliqué au train et au test. [file :3][file :4]

```
missing_cols = ['temperature', 'pouls', 'oxygene']

imputer = SimpleImputer(strategy='mean')

# Apprentissage des paramètres d'imputation sur le train
imputer.fit(X_train[missing_cols])

# Application de l'imputation sur train et test
X_train[missing_cols] = imputer.transform(X_train[missing_cols])
X_test[missing_cols] = imputer.transform(X_test[missing_cols])
```

[file :4]

Cette méthode suit les recommandations méthodologiques en séparant clairement les phases d'apprentissage et de test pour toutes les transformations basées sur les données. [file :3]

5 Construction du modèle de classification

5.1 Choix du modèle : régression logistique

Le modèle retenu est une régression logistique binaire, bien adaptée à une cible de type 0/1 et offrant une interprétabilité satisfaisante grâce à ses coefficients. [file :3][file :4][file :5]

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, Y_train)
```

[file :4]

5.2 Prédictions sur le jeu de test

Les prédictions sont ensuite calculées sur l'ensemble de test. [file :4]

```
y_pred = model.predict(X_test)
```

[file :4]

6 Évaluation des performances

6.1 Accuracy et matrice de confusion

L'accuracy globale du modèle sur le jeu de test est calculée à l'aide de la fonction `accuracy_score`. [file :4]

```
acc = accuracy_score(Y_test, y_pred)
print("Accuracy du modèle :", acc)
```

[file :4]

Le score obtenu est proche de 0,996, ce qui correspond à un taux d'erreur très faible sur l'échantillon de test. [file :4]

La matrice de confusion permet de détailler la nature des erreurs (faux positifs et faux négatifs). [file :3][file :4]

```
cm = confusion_matrix(Y_test, y_pred)
print(cm)

plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='summer')
plt.title('Matrice de confusion')
plt.ylabel('Vraie classe')
plt.xlabel('Classe prédite')
plt.show()
```

[file :4]

Cette matrice distingue clairement les vrais positifs (malades correctement prédits), les vrais négatifs (sains correctement prédits), ainsi que les faux positifs et faux négatifs, ces derniers étant les plus sensibles en contexte médical. [file :3][file :4]

6.2 Précision, rappel et F1-score

Un rapport de classification détaillé est généré pour obtenir la précision, le rappel et le F1-score de chaque classe. [file :3][file :4]

```
print(classification_report(Y_test, y_pred, target_names=['sain', 'malade']))
```

[file :4]

En pratique, l'accent est mis sur le rappel de la classe « malade », afin de minimiser les cas de patients malades non détectés par le modèle, en accord avec les exigences de sécurité clinique décrites dans la correction. [file :3]

7 Visualisations complémentaires

7.1 Température moyenne selon l'état de santé

Pour mieux comprendre le comportement des variables clés, des barplots comparent les moyennes par classe. [file :4][file :5]

```
plt.figure(figsize=(5, 4))
sns.barplot(data=data, x='label', y='temperature',
            palette=['red', 'blue'])
plt.title('Température moyenne selon le label')
plt.xlabel('label (0 = sain, 1 = malade)')
plt.ylabel('Température')
plt.show()
```

[file :4]

Ces graphiques montrent notamment une température moyenne plus élevée chez les patients malades, ce qui est cohérent avec la présence de fièvre. [file :4][file :5]

7.2 Saturation en oxygène moyenne selon l'état de santé

```
plt.figure(figsize=(5, 4))
sns.barplot(data=data, x='label', y='oxygene',
            palette=['violet', 'cyan'])
plt.title("Saturation en oxygène moyenne selon le label")
plt.xlabel('label (0 = sain, 1 = malade)')
plt.ylabel('Oxygène')
plt.show()
```

[file :4]

On observe également des différences de saturation en oxygène entre patients sains et malades, ce qui renforce la plausibilité clinique du modèle. [file :4][file :5]

8 Conclusion

Ce projet illustre l'ensemble du cycle de vie d'un projet de data science appliqué à la santé : analyse exploratoire, nettoyage des données (gestion des valeurs aberrantes et manquantes), préparation des variables, séparation train/test, entraînement d'un modèle de régression logistique et évaluation à l'aide de métriques adaptées. [file :3][file :4][file :5]

Le modèle atteint une accuracy d'environ 99,6 % sur le jeu de test, avec un nombre très limité de faux négatifs, ce qui est essentiel dans un contexte où rater un patient malade est particulièrement dangereux. [file :4] En suivant les bonnes pratiques méthodologiques mises en avant dans la correction (notamment la prévention du *data leakage* et le choix de métriques cliniquement pertinentes), ce travail montre comment transformer un simple tableau de mesures physiologiques en un assistant de décision fiable pour la détection d'une maladie. [file :3][file :4][file :5]