Rapport de projet

Prédiction de la maladie cardiaque par régression logistique et forêt aléatoire

---

1. Introduction

La maladie cardiaque représente une cause majeure de mortalité dans le monde. La détection précoce de patients à risque constitue donc un enjeu important, tant pour la prévention que pour la prise en charge. Dans ce contexte, les méthodes de **classification supervisée** peuvent être utilisées pour prédire la présence d'une maladie cardiaque à partir de caractéristiques cliniques mesurées chez les patients.

Le présent projet a pour objectif de construire un modèle de prédiction binaire indiquant, pour chaque individu, s'il est susceptible de présenter une maladie cardiaque (`target = 1`) ou non (`target = 0`). Pour cela, deux modèles de machine learning ont été mis en œuvre et comparés :

- une **régression logistique**, modèle linéaire probabiliste largement utilisé en statistique médicale ;
- une **forêt aléatoire** (*Random Forest*), modèle d'ensemble basé sur de multiples arbres de décision.

Le programme développé réalise les étapes suivantes :

1. chargement et exploration d'un jeu de données de maladie cardiaque ;
2. séparation du jeu de données en ensembles d'entraînement, de validation et de test ;
3. prétraitement des variables explicatives ;
4. entraînement des deux modèles ;
5. évaluation comparative des performances ;
6. sélection du meilleur modèle et évaluation finale sur l'ensemble de test.

---

2. Description du jeu de données

Le jeu de données utilisé est un dataset de maladie cardiaque contenant **1025 observations** et **14 colonnes**. Il comprend :

- **13 variables explicatives (features)** :
  - `age` : âge du patient ;
  - `sex` : sexe (0 = femme, 1 = homme) ;
  - `cp` : type de douleur thoracique ;
  - `trestbps` : pression artérielle au repos (en mm Hg) ;
  - `chol` : taux de cholestérol sanguin (mg/dl) ;
  - `fbs` : glycémie à jeun > 120 mg/dl (0 ou 1) ;
  - `restecg` : résultats de l'électrocardiogramme au repos ;
  - `thalach` : fréquence cardiaque maximale atteinte ;
  - `exang` : angine de poitrine induite par l'effort (0 ou 1) ;

    - `oldpeak` : dépression du segment ST induite par l'effort par rapport au repos ;
    - `slope` : pente du segment ST au pic de l'exercice ;
    - `ca` : nombre de vaisseaux majeurs colorés par fluoroscopie ;
    - `thal` : type de thalassémie (catégorielle codée en entier) ;
- **1 variable cible** :
    - `target` : 0 = absence de maladie cardiaque, 1 = présence de maladie cardiaque.

Après chargement, une première exploration du dataset est effectuée :

- l'aperçu des premières lignes (via `df.head()`) permet de vérifier la cohérence des valeurs ;
- `df.info()` affiche le nombre d'observations, les types de chaque colonne et confirme l'absence de valeurs manquantes ;
- `df.describe()` fournit des statistiques descriptives (moyenne, écart-type, minimum, maximum, quartiles) pour chaque variable numérique.

La répartition de la variable cible est la suivante :

- `target = 1` (présence de maladie cardiaque) : 526 cas,
- `target = 0` (absence de maladie cardiaque) : 499 cas.

Le dataset est donc **assez équilibré** : les deux classes sont présentes en proportions comparables. Ce point est important, car il évite d'avoir un modèle fortement biaisé vers une seule classe.

Toutes les colonnes sont de type **numérique** (entiers ou flottants), ce qui simplifie le prétraitement et permet d'utiliser directement un scaler standard sans encodage supplémentaire.

---

3. Méthodologie et préparation des données

3.1. Séparation des variables explicatives et de la cible

La première étape de la préparation consiste à séparer :

- les variables explicatives **X** : toutes les colonnes sauf `target` ;
- la variable cible **y** : la colonne `target`, convertie en entier.

Cette séparation est indispensable pour entraîner les modèles de classification, qui apprennent une fonction de décision $f : X \to {0,1}$ à partir de paires (X, y).

Une liste des noms de features (`feature_names`) est également extraite, afin de pouvoir réutiliser ces informations, notamment pour l'interprétation des modèles.

3.2. Découpage en ensembles d'entraînement, de validation et de test

Pour évaluer correctement les performances des modèles et éviter les biais, le jeu de données est découpé en trois sous-ensembles :

- **Ensemble d'entraînement (train)** : utilisé pour ajuster les paramètres internes des modèles ;
- **Ensemble de validation (validation)** : utilisé pour comparer les modèles entre eux et choisir le

meilleur ;
- **Ensemble de test (test)** : utilisé uniquement à la fin, pour mesurer la performance réelle du modèle sélectionné sur des données jamais vues.

Le découpage est réalisé à l'aide de la fonction `train_test_split` de scikit-learn, en deux étapes :

1. séparation en :
    - 85 % pour train + validation,
    - 15 % pour test ;
2. découpage de la partie train + validation en :
    - 70 % train,
    - 15 % validation.

La **stratification** est activée (`stratify=y`) pour conserver la même proportion de classes dans chaque sous-ensemble.

Au final, les tailles obtenues sont :

- **train** : 717 observations,
- **validation** : 154 observations,
- **test** : 154 observations.

Ce schéma permet de sélectionner un modèle à partir de la performance sur validation, puis de contrôler sa **généralisation** sur le test.

3.3. Prétraitement des variables

Même si toutes les colonnes sont numériques, leurs échelles sont différentes (par exemple, `age` varie entre 29 et 77, alors que `oldpeak` est une variable continue sur une autre échelle). Certains modèles, comme la Régression Logistique, sont sensibles à l'échelle des variables.

Afin d'homogénéiser les données, un prétraitement de type **standardisation** est appliqué :

- utilisation de `StandardScaler` pour centrer chaque feature (moyenne 0) et la réduire (variance 1) ;
- application de ce scaler à l'ensemble des 13 variables explicatives.

Dans le programme, ce prétraitement est intégré à un `ColumnTransformer`, puis placé en première étape dans les pipelines des deux modèles. De cette manière, la même transformation est automatiquement appliquée aux données d'entraînement, de validation et de test, garantissant la cohérence du traitement.

---

4. Description détaillée du programme

Le programme est écrit en Python et s'appuie principalement sur les bibliothèques **NumPy**, **pandas**, **matplotlib** et **scikit-learn**.

4.1. Imports et configuration initiale

En première partie, le script importe :

- `numpy` et `pandas` pour la manipulation des données ;
- `matplotlib.pyplot` pour la visualisation (principalement les courbes ROC et les graphiques d'importance de variables) ;
- les fonctions de **scikit-learn** nécessaires :
  - `train_test_split` pour le découpage des données,
  - `StandardScaler`, `Pipeline`, `ColumnTransformer` pour le prétraitement,
  - `LogisticRegression`, `RandomForestClassifier` pour les modèles,
  - les métriques : `accuracy_score`, `precision_score`, `recall_score`, `f1_score`, `classification_report`, `confusion_matrix`, `roc_curve`, `roc_auc_score`.

Une constante `RANDOM_STATE = 42` est définie et utilisée partout afin de garantir la **reproductibilité** des résultats (mêmes splits, mêmes modèles, mêmes performances à chaque exécution).

4.2. Chargement et analyse exploratoire des données

Le script charge ensuite le fichier CSV `heart.csv` stocké dans un dossier de données (par exemple `data/heart.csv`) à l'aide de `pandas.read_csv`.

Immédiatement après le chargement, plusieurs affichages sont effectués :

- `df.head()` : aperçu des 5 premières lignes pour vérifier visuellement la structure ;
- `df.info()` : nombre de lignes, nombre de colonnes, types des colonnes, présence ou non de valeurs manquantes ;
- `df.describe()` : statistiques descriptives sur les colonnes numériques ;
- `df['target'].value_counts()` : répartition de la variable cible entre les classes 0 et 1.

Ces affichages permettent de documenter le dataset directement dans la console au moment de l'exécution du programme.

4.3. Construction de X et y, et découpage des données

Le programme construit :

- **X** en supprimant la colonne `target` du DataFrame ;
- **y** en récupérant la colonne `target` et en la convertissant en type entier.

Les noms des features sont stockés dans la variable `feature_names`, qui sera réutilisée plus tard, notamment pour les graphiques d'importance de variables.

Ensuite, le découpage en trois ensembles est réalisé :

1. appel à `train_test_split` pour séparer un ensemble de test représentant 15 % des données initiales (avec `stratify=y` et `random_state=RANDOM_STATE`) ;
2. nouveau `train_test_split` sur la partie restante pour constituer l'ensemble de validation (15 % au total), de manière à obtenir la répartition finale 70 % / 15 % / 15 %.

Les dimensions des ensembles obtenus sont affichées :

- taille de `X_train`,
- taille de `X_val`,
- taille de `X_test`.

Cela permet de vérifier rapidement que le découpage s'est déroulé comme prévu.

4.4. Création du préprocesseur

Le script définit ensuite le **préprocesseur** des données :

- toutes les colonnes de `X` sont considérées comme des variables numériques ;
- un pipeline `numeric_transformer` applique un `StandardScaler` à ces colonnes ;
- un `ColumnTransformer` est créé pour appliquer ce pipeline à la liste `numeric_features`.

Ce préprocesseur encapsule toute la logique de mise à l'échelle des données et sera réutilisé par les deux modèles.

L'utilisation du `ColumnTransformer` garantit que le prétraitement est entièrement intégré dans le pipeline de scikit-learn : il est appliqué automatiquement à l'entraînement (`fit`) et à la prédiction (`predict` / `predict_proba`).

4.5. Définition des modèles et pipelines

Deux pipelines de modèles sont ensuite définis :

1. **Pipeline de Régression Logistique**

   ```
   logreg_clf = Pipeline(steps=[
       ("preprocess", preprocessor),
       ("model", LogisticRegression(max_iter=1000, random_state=RANDOM_STATE))
   ])
   ```

   Ce pipeline applique d'abord le préprocesseur (StandardScaler), puis entraîne une régression logistique sur les données standardisées.

2. **Pipeline de Forêt Aléatoire**

   ```
   rf_clf = Pipeline(steps=[
       ("preprocess", preprocessor),
       ("model", RandomForestClassifier(
           n_estimators=200,
           max_depth=None,
           random_state=RANDOM_STATE
       ))
   ])
   ```

   De même, ce pipeline applique d'abord le préprocesseur, puis entraîne une forêt aléatoire à 200 arbres.

Les deux modèles sont réunis dans un dictionnaire `models` :

```
models = {
    "Logistic Regression": logreg_clf,
    "Random Forest": rf_clf
}
```

Ce dictionnaire permet de boucler facilement sur les deux pipelines lors de la phase d'évaluation.

4.6. Fonction d'évaluation `evaluate_model`

Une fonction dédiée, `evaluate_model`, est utilisée pour entraîner et évaluer chaque modèle sur l'ensemble de validation. Elle prend en entrée :

- un nom de modèle (`name`) ;
- le pipeline de modèle (`model`) ;
- les ensembles `X_train`, `y_train`, `X_val`, `y_val`.

La fonction réalise les opérations suivantes :

1. **Affichage d'un en-tête** indiquant le nom du modèle ;
2. **Entraînement** du pipeline avec `model.fit(X_train, y_train)` :
   le préprocesseur est ajusté sur les données d'entraînement, puis le modèle est entraîné ;
3. **Prédictions** sur l'ensemble de validation :
   - `y_pred = model.predict(X_val)` pour les classes prédites ;
   - `y_proba = model.predict_proba(X_val)[:, 1]` si le modèle dispose de la méthode `predict_proba` (ce qui est le cas ici) ;
4. **Calcul des métriques principales** :
   - Accuracy,
   - Précision,
   - Rappel,
   - F1-score ;
5. **Affichage du classification report** (`classification_report`) : détail des métriques par classe (0 et 1) ;
6. **Affichage de la matrice de confusion** (`confusion_matrix`) : tableau indiquant le nombre de vrais positifs, faux positifs, vrais négatifs et faux négatifs ;
7. **Calcul et tracé de la courbe ROC**, si les probabilités `y_proba` sont disponibles :
   - calcul de `fpr`, `tpr` et des seuils via `roc_curve` ;
   - calcul de l'aire sous la courbe (AUC) via `roc_auc_score` ;
   - affichage de la courbe ROC avec `matplotlib`.

La fonction renvoie un dictionnaire contenant le nom du modèle, le pipeline entraîné, et les métriques calculées (accuracy, précision, rappel, F1).

4.7. Boucle d'entraînement et sélection du meilleur modèle

Après la définition de la fonction `evaluate_model`, le script parcourt le dictionnaire `models` :

- pour chaque paire (nom, pipeline), il appelle `evaluate_model` avec les ensembles d'entraînement et de validation ;
- les résultats sont stockés dans une liste `results_val`.

Une fois tous les modèles évalués, le script recherche le **meilleur modèle** sur la validation en utilisant le **F1-score** comme critère :

```
best_model = max(results_val, key=lambda r: r["f1"])
```

Le nom du meilleur modèle et sa valeur de F1-score sont affichés. Le pipeline correspondant (`final_model`) est ensuite utilisé pour évaluer les performances sur l'ensemble de test.

4.8. Évaluation finale sur l'ensemble de test

L'évaluation finale du meilleur modèle se fait sur `X_test` et `y_test` :

- prédiction des classes (`y_test_pred`) ;
- prédiction des probabilités (`y_test_proba`) si disponible ;
- calcul des métriques : accuracy, précision, rappel, F1-score ;
- affichage du classification report et de la matrice de confusion ;
- calcul et tracé de la courbe ROC et de l'AUC sur le test.

Cette étape fournit une **estimation réaliste de la performance du modèle** sur des données nouvelles, non utilisées pendant l'entraînement ni pour le choix du modèle.

4.9. Analyse de l'importance des variables

Enfin, le script réalise une analyse de l'importance des variables à partir du modèle de base :

- si le modèle final est une **Forêt Aléatoire**, l'attribut `feature_importances_` est utilisé :
    - les importances sont triées par ordre décroissant ;
    - pour chaque feature, son importance est affichée dans la console ;
    - un graphique en barres est tracé pour visualiser les importances des variables ;
- si le modèle final est une **Régression Logistique**, le vecteur de coefficients `coef_` est utilisé :
    - les coefficients sont triés selon leur valeur absolue ;
    - pour chaque variable, le coefficient est affiché ;
    - un graphique en barres montre la contribution (positive ou négative) de chaque feature.

Cette analyse permet de mieux comprendre le rôle des différentes variables dans la prédiction de la maladie cardiaque.

---

5. Modèles de classification

5.1. Régression logistique

La régression logistique est un modèle linéaire de classification binaire qui modélise la probabilité d'appartenance à la classe positive à l'aide de la fonction sigmoïde :

$$P(Y = 1 \mid X = x) = \sigma(w^\top x + b), \quad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Les paramètres w et b sont estimés en minimisant une fonction de perte de type entropie croisée (log-loss). La décision de classification est ensuite obtenue en appliquant un seuil (généralement 0,5) à la probabilité prédite.

La régression logistique présente plusieurs avantages :

- interprétabilité des coefficients (effet de chaque feature sur la log-odds de la maladie) ;
- rapidité d'entraînement ;
- bonnes performances sur des problèmes où la frontière de décision est globalement linéaire.

Elle peut cependant être limitée dans le cas de relations fortement non linéaires ou d'interactions complexes entre variables.

Dans le projet, le modèle utilisé est :

```
LogisticRegression(max_iter=1000, random_state=RANDOM_STATE)
```

5.2. Forêt aléatoire (Random Forest)

La forêt aléatoire est un modèle d'ensemble qui combine un grand nombre d'arbres de décision, chacun entraîné sur un échantillon bootstrap des données d'entraînement, avec une sélection aléatoire de features à chaque nœud.

Le principe est le suivant :

- chaque arbre est entraîné sur un sous-échantillon des données (bagging) ;
- à chaque nœud, seule une partie des variables est testée pour trouver la meilleure coupure ;
- la prédiction finale est obtenue par vote majoritaire (classification) ou moyennage (régression).

Cette architecture permet de :

- réduire la variance par rapport à un arbre unique ;
- capturer des relations non linéaires et des interactions entre features ;
- fournir une mesure d'importance des variables.

Le modèle utilisé dans le projet est :

```
RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=RANDOM_STATE
)
```

6. Résultats expérimentaux

6.1. Performances sur l'ensemble de validation

Les performances obtenues sur l'ensemble de validation sont :

- **Régression logistique** :
  - Accuracy ≈ 0,792
  - Précision ≈ 0,764
  - Rappel ≈ 0,861
  - F1-score ≈ 0,810
  - AUC-ROC ≈ 0,895
- **Forêt aléatoire** :
  - Accuracy ≈ 0,994
  - Précision ≈ 0,988
  - Rappel ≈ 1,000
  - F1-score ≈ 0,994
  - AUC-ROC ≈ 0,999

La forêt aléatoire surpasse très largement la régression logistique sur l'ensemble des métriques.

6.2. Sélection du modèle

Le critère de sélection retenu est le **F1-score** sur l'ensemble de validation. Selon ce critère, la forêt aléatoire est choisie comme modèle final.

6.3. Performances sur l'ensemble de test

Sur l'ensemble de test, le modèle de forêt aléatoire présente les performances suivantes :

- Accuracy (test) ≈ 0,987
- Précision (test) ≈ 0,975
- Rappel (test) ≈ 1,000
- F1-score (test) ≈ 0,988
- AUC-ROC (test) ≈ 0,999

Les résultats sur test étant très proches de ceux obtenus sur validation, la capacité de généralisation du modèle apparaît satisfaisante.

---

7. Conclusion

Le projet met en œuvre un système de classification supervisée pour la **prédiction de la maladie cardiaque** à partir de variables cliniques.

Les principales étapes ont été :

- l'exploration d'un dataset de 1025 patients, avec 13 features numériques et une cible binaire ;
- la préparation et la normalisation des données ;
- l'entraînement et la comparaison de deux modèles : **régression logistique** et **forêt aléatoire** ;
- l'évaluation rigoureuse sur des ensembles de validation et de test.

Les résultats montrent que la **forêt aléatoire** offre des performances nettement supérieures à celles de la régression logistique sur ce jeu de données, avec des métriques très élevées (F1-score ≈ 0,988, AUC-ROC ≈ 0,999 sur le test).

Des améliorations possibles incluent l'ajustement des hyperparamètres par validation croisée, la comparaison avec d'autres méthodes d'ensemble (boosting), et une analyse plus détaillée de l'importance des variables pour interpréter finement les facteurs de risque de la maladie cardiaque.