Breast Cancer Prediction

Mini-Projet en IA

IWMBigData

CHAABANI ABDELILAH - EL AZZOUZI AYOUB

Sommaire :

I. Contexte

Le cancer du sein est l'un des cancers les plus courants chez les femmes dans le monde entier. La détection précoce de la plupart des cancers du sein est cruciale pour améliorer les chances de traitement et de survie des patientes. L'objectif de ce projet est de développer un modèle de machine learning capable de prédire si une tumeur est bénigne ou maligne en se basant sur des caractéristiques médicales mesurées. Nous avons utilisé divers algorithmes de machine learning pour évaluer leurs performances et déterminer le meilleur modèle pour cette tâche.

II. Le choix des technologies

Pour ce projet, nous avons choisi les technologies suivantes :

- **Python :** Langage de programmation important utilisé pour le développement des modèles.
- **Pandas :** Bibliothèque utilisée pour la manipulation et l'analyse des données.
- **scikit-learn** : Bibliothèque de machine learning utilisée pour la création et l'évaluation des modèles.
- **Matplotlib** : Bibliothèque utilisée pour la visualisation des données.
- **Google Colab** : Environnement de développement intégré pour exécuter le code Python et visualiser les résultats.

III. Dataset

Le dataset utilisé dans ce projet est le "Breast Cancer DataSet". Il contient 569 échantillons de tumeurs diagnostiquées avec les caractéristiques suivantes :

- 30 caractéristiques de la tumeur (par exemple, la taille, la texture, la symétrie, etc.).
- La variable cible est le diagnostic (bénin ou malin).
- Les données ont été prétraitées pour encoder la variable cible et normaliser les caractéristiques.

Les premières étapes incluent le chargement des bibliothèques nécessaires, l'importation des données, le prétraitement, et la séparation des données en ensembles d'entraînement et de test.

Chargement des bibliothèques

```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder

    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

    from sklearn.svm import SVC

    import xgboost as xgb
    from xgboost import XGBClassifier

    from sklearn.ensemble import RandomForestClassifier

    from sklearn.ensemble import GradientBoostingClassifier

    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
    import matplotlib.pyplot as plt
```

Chargement des données

```
[ ] data = pd.read_csv('/content/data.csv')
```

Exploration des données

```
[ ] # Afficher les premières lignes du dataframe
    print("Les premières lignes du dataframe:\n")
    print(data.head())

    # Afficher des informations sur le dataframe
    print("\nInformations sur le dataframe:\n")
    print(data.info())

    # Décrire les statistiques des colonnes numériques
    print("\nStatistiques descriptives des colonnes numériques:\n")
    print(data.describe())
```

Prétraitement des données

```
[ ]  # Suppression la colonne 'id'
     data = data.drop(columns=['id'])

     # Conversion la colonne 'diagnosis' en valeurs numériques
     label_encoder = LabelEncoder()
     data['diagnosis'] = label_encoder.fit_transform(data['diagnosis'])

     # Séparation les caractéristiques et la cible
     X = data.drop(columns=['diagnosis'])
     y = data['diagnosis']

     # Division des données en ensembles d'entraînement et de test
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

IV. Méthodes proposées

Nous avons implémenté et évalué plusieurs algorithmes de machine learning pour la classification binaire (bénin vs. Malin). Les algorithmes suivants ont été utilisés :

1. **Decision Tree**
   Description : Les arbres de décision sont des modèles prédictifs qui utilisent une série de règles de décision pour classer les données. Chaque nœud de l'arbre représente une caractéristique (attribut) des données, et chaque branche représente une règle de décision basée sur cette caractéristique. Les feuilles de l'arbre représentent les classes de sortie.

   Avantages :

   - Facile à interpréter et visualiser.
   - Peut gérer des données catégorielles et numériques.
   - Ne nécessite pas de normalisation des données.

   Inconvénients :

   - Prône à l'overfitting si l'arbre est trop profond.
   - Les petites variations dans les données peuvent entraîner des arbres très différents.

2. **Support Vector Machine (SVM)**
   Description : SVM est un algorithme de class qui cherche à trouver l'hyperplan optimal qui sépare les classes dans un espace à haute dimension. Les vecteurs de support sont les points de données qui définissent les marges du classificateur.

Avantages :

- Efficace dans des espaces à haute dimension.
- Efficace lorsque le nombre de dimensions est supérieur au nombre d'échantillons.
- Utilise un sous-ensemble de points d'apprentissage (vecteurs de support), ce qui le rend efficace en termes de mémoire.

Inconvénients :

- Peut-être inefficace pour des datasets très larges.
- La sélection du noyau approprié peut être complexe.
- Sensible aux choix des paramètres et du noyau.

3. **XGBoost**
   Description : XGBoost (Extreme Gradient Boosting) est une implémentation optimisée des arbres de décision en gradient boosting. Il construit les modèles de manière séquentielle, où chaque nouveau modèle corrige les erreurs du modèle précédent.

   Avantages :

   - Haute performance et efficacité.
   - Capable de gérer des données manquantes.
   - Prend en charge la régularisation pour éviter l'overfitting.
   - Efficace en termes de mémoire et de calcul.

   Inconvénients :

   - Complexité accrue par rapport aux arbres de décision simples.
   - Nécessite un réglage minutieux des hyperparamètres.

4. **Random Forest**
   Description : Random Forest est un ensemble d'arbres de décision formés sur des sous-échantillons du dataset. Les prédictions des arbres individuels sont agrégées (par exemple, par vote) pour améliorer la précision et contrôler l'overfitting.

   Avantages :

   - Réduit l'overfitting par rapport aux arbres de décision simples.
   - Peut gérer les valeurs manquantes et maintenir l'exactitude pour une grande proportion des données manquantes.

- Robuste aux variations dans les données.

Inconvénients :

- Moins interprétable qu'un seul arbre de décision.
- Nécessite plus de ressources en termes de calcul et de mémoire.

5. **Gradient Boosting Decision Trees (GBDT)**
   Description : GBDT est une méthode d'ensemble qui construit des modèles de prédiction sous forme d'une série d'arbres de décision. Chaque arbre corrige les erreurs du modèle précédent, améliorant ainsi progressivement la précision du modèle global.

   Avantages :

   - Excellente performance pour une variété de tâches de classification et de régression.
   - Capable de gérer les interactions complexes entre les caractéristiques.
   - Efficace pour les données de haute dimension.

   Inconvénients :

   - Peut-être lent à entraîner en raison de sa nature séquentielle.
   - Prône à l'overfitting si le nombre d'arbres est trop élevé.

6. **Linear Regression**
   Description : La régression linéaire est une méthode de modélisation de la relation entre une variable dépendante et une ou plusieurs variables indépendantes en ajustant une ligne de meilleure adéquation. Pour la classification, les prédictions continues sont souvent converties en classes binaires en utilisant un seuil (par exemple, 0.5).

   Avantages :

   - Simple à implémenter et interpréter.
   - Rapide à entraîner.
   - Fonctionne bien lorsque la relation entre les variables indépendantes et dépendantes est linéaire.

   Inconvénients :

   - Peut-être inapproprié pour des relations non linéaires.
   - Moins performant pour des problèmes de type complexes.

Pour chaque algo, nous avons suivi les étapes suivantes :

- **Prétraitement des données :** Suppression des colonnes inutiles, encodage de la Variable cible, et séparation des données en ensembles d'entraînement et de test.
- **Entraînement du modèle :** Création et entraînement du modèle sur l'ensemble d'entraînement.
- **Prédiction :** Utilisation du modèle pour prédire les étiquettes sur l'ensemble de test.
- **Évaluation :** Calcul des métriques de performance telles que l'accuracy, la matrice de confusion, et le rapport de classification.

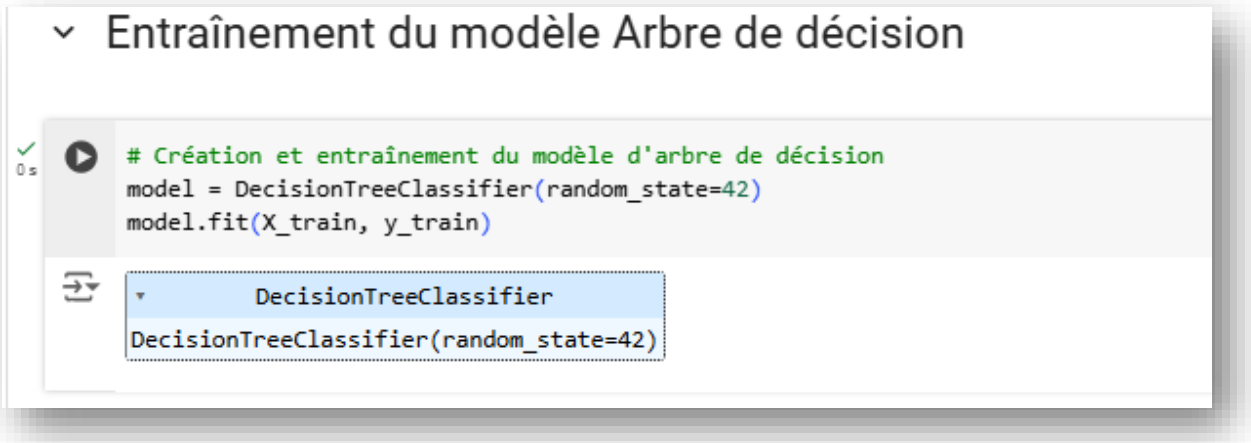
V. Evaluation des matrices

Les performances des modèles ont été évaluées à l'aide des métriques suivantes :

- **Accuracy :** Mesure la proportion de prédictions correctes.
- **Matrice de confusion :** Fournit une vue détaillée des performances en montrant les vrais positifs, les faux positifs, les vrais négatifs, et les faux négatifs.
- **Rapport de classification :** Inclut la précision, le rappel et le score F1 pour chaque classe.

Voici un résumé des résultats pour chaque modèle :

1. **Decision Tree**



Évaluation du modèle Arbre de décision

```
[8]  # Faire des prédictions sur l'ensemble de test
     y_pred = model.predict(X_test)

     # Évaluer les performances du modèle
     accuracy = accuracy_score(y_test, y_pred)
     conf_matrix = confusion_matrix(y_test, y_pred)
     class_report = classification_report(y_test, y_pred)

     print("Accuracy:", accuracy)
     print("Confusion Matrix:\n", conf_matrix)
     print("Classification Report:\n", class_report)
```

```
Accuracy: 0.9473684210526315
Confusion Matrix:
 [[68  3]
 [ 3 40]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        71
           1       0.93      0.93      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

2. SVM

Création et entraînement du modèle SVM

```
# Création et entraînement du modèle SVM
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
```

```
SVC(kernel='linear', random_state=42)
```

Prédictions avec le modèle SVM

```
#Prédictions sur l'ensemble de test
svm_y_pred = svm_model.predict(X_test)
print(svm_y_pred)
```

```
[0 1 1 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1
 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 0 0 1 0 0 1
 0 1 1]
```

Évaluation du modèle SVM

```
# Évaluation du modèle
svm_accuracy = accuracy_score(y_test, svm_y_pred)
svm_conf_matrix = confusion_matrix(y_test, svm_y_pred)
svm_class_report = classification_report(y_test, svm_y_pred)

print("SVM Accuracy:", svm_accuracy)
print("SVM Confusion Matrix:\n", svm_conf_matrix)
print("SVM Classification Report:\n", svm_class_report)
```

```
SVM Accuracy: 0.956140350877193
SVM Confusion Matrix:
 [[70  1]
 [ 4 39]]
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        71
           1       0.97      0.91      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

3. XGBoost

Création et entraînement du modèle XGBoost

```
# Création et entraînement du modèle XGBoost
xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

Prédictions avec le modèle XGBoost

```
# Prédictions sur l'ensemble de test
xgb_y_pred = xgb_model.predict(X_test)
print(xgb_y_pred)
```

```
[0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0 1
 0 0 1]
```

Évaluation du modèle XGBoost

```
# Évaluation du modèle XGBoost
xgb_accuracy = accuracy_score(y_test, xgb_y_pred)
xgb_conf_matrix = confusion_matrix(y_test, xgb_y_pred)
xgb_class_report = classification_report(y_test, xgb_y_pred)

print("XGBoost Accuracy:", xgb_accuracy)
print("XGBoost Confusion Matrix:\n", xgb_conf_matrix)
print("XGBoost Classification Report:\n", xgb_class_report)
```

```
XGBoost Accuracy: 0.956140350877193
XGBoost Confusion Matrix:
 [[69  2]
 [ 3 40]]
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        71
           1       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

4. Random Forest

Création et entraînement du modèle Random Forest

```
# Création et entraînement du modèle Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

Prédictions avec le modèle Random Forest

```
# Prédictions sur l'ensemble de test
rf_y_pred = rf_model.predict(X_test)
print(rf_y_pred)
```

```
[0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1
 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 0 0 1 0 0 1
 0 0 1]
```

Évaluation du modèle Random Forest

```
# Évaluation du modèle Random Forest
rf_accuracy = accuracy_score(y_test, rf_y_pred)
rf_conf_matrix = confusion_matrix(y_test, rf_y_pred)
rf_class_report = classification_report(y_test, rf_y_pred)

print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest Confusion Matrix:\n", rf_conf_matrix)
print("Random Forest Classification Report:\n", rf_class_report)
```

```
Random Forest Accuracy: 0.9649122807017544
Random Forest Confusion Matrix:
 [[70  1]
 [ 3 40]]
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        71
           1       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

5. GBDT

Création et entraînement du modèle GBDT

```
# Création et entraînement du modèle GBDT
gbdt_model = GradientBoostingClassifier(random_state=42)
gbdt_model.fit(X_train, y_train)
```

```
GradientBoostingClassifier(random_state=42)
```

Prédictions avec le modèle GBDT

```
# Prédictions sur l'ensemble de test
gbdt_y_pred = gbdt_model.predict(X_test)
print(gbdt_y_pred)
```

```
[0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1
 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0 1
 0 0 1]
```

Évaluation du modèle GBDT

```
#Évaluation du modèle GBDT
gbdt_accuracy = accuracy_score(y_test, gbdt_y_pred)
gbdt_conf_matrix = confusion_matrix(y_test, gbdt_y_pred)
gbdt_class_report = classification_report(y_test, gbdt_y_pred)

print("GBDT Accuracy:", gbdt_accuracy)
print("GBDT Confusion Matrix:\n", gbdt_conf_matrix)
print("GBDT Classification Report:\n", gbdt_class_report)
```

```
GBDT Accuracy: 0.956140350877193
GBDT Confusion Matrix:
 [[69  2]
 [ 3 40]]
GBDT Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        71
           1       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

6. Linear Regression

Création et entraînement du modèle de régression linéaire

```
# Création et entraînement du modèle de régression linéaire
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

Prédictions avec le modèle de régression linéaire

```
# Prédictions avec le modèle de régression linéaire
y_pred = lr_model.predict(X_test)
print(y_pred)
```

```
[ 0.37831645  0.82196147  0.72487932 -0.1320503  -0.11698318  1.66303713
  1.07355607  0.55787645  0.71914237  0.03639739  0.205858    0.66807416
  0.09518243  0.60333496  0.0663303   0.82249825  0.08986729 -0.19445658
 -0.41615414  0.95535282  0.34958323  0.1247244   1.38792384 -0.08061015
  0.03784724  0.15321053  0.00192859  0.12950561  0.05613743  1.16893149
 -0.0238613  -0.04190872  0.18395555  0.0843411  -0.07186427  0.15710955
  0.51253374  0.37624738  0.74631453  0.23157771 -0.15259719  0.96200807
  0.02873487  0.00226268  0.44839943  0.34412526  0.08171919 -0.02580964
  0.19472146 -0.01543748  0.79025547  1.07875586  0.44157032  0.40962104
 -0.02323689  0.21427041  0.06040322  1.36416183  0.44152119 -0.04937824
  0.05593734  1.1953921   1.35465216  0.19223021  0.09791508  0.41690135
  1.16982229  1.15422403  0.02122549  0.16565316  0.90876926  0.90468069
  0.1513417   0.74128644  0.0050424   0.16438241  0.19795219  0.41424796
 -0.17901275  0.24233827  0.59003867 -0.05654014  0.46968994  1.13159216
  0.6808122   0.85938375  1.36596822  0.96142258  0.01963916  0.05721644
  0.17592452  0.20805153  0.19981479 -0.10421683  0.01015646  0.0434749
  1.00308431  1.19926989 -0.03364721  0.86955229  0.75442703 -0.27709941
  0.78905217  0.84266614  0.23600559  0.25878762  0.19636649  1.10971523
  0.291374    0.25258836  0.83843779  0.03208652  0.43378739  0.86696411]
```

Conversion des prédictions en classes binaires

```
# Convertion des prédictions en classes binaires (0 ou 1)
y_pred_binary = [1 if pred > 0.5 else 0 for pred in y_pred]
print(y_pred_binary)
```

```
[0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0
```

Évaluation du modèle de régression linéaire

```
#Évaluation du modèle de régression linéaire
lr_accuracy = accuracy_score(y_test, y_pred_binary)
lr_conf_matrix = confusion_matrix(y_test, y_pred_binary)
lr_class_report = classification_report(y_test, y_pred_binary)

print("Linear Regression Accuracy:", lr_accuracy)
print("Linear Regression Confusion Matrix:\n", lr_conf_matrix)
print("Linear Regression Classification Report:\n", lr_class_report)
```

```
Linear Regression Accuracy: 0.956140350877193
Linear Regression Confusion Matrix:
 [[70  1]
 [ 4 39]]
Linear Regression Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        71
           1       0.97      0.91      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

VI. Conclusion

Tous les modèles testés ont montré des performances élevées dans la classification des tumeurs en bénignes et malignes.

Le modèle **Random Forest** a obtenu le meilleur résultat en termes d'accuracy avec un taux de précision de **96%**. Ce modèle est recommandé pour la tâche de classification du cancer du sein dans ce dataset.

Les performances des modèles montrent que les techniques de machines learning peuvent être très efficaces pour aider à la détection précoce du cancer du sein, ce qui peut potentiellement sauver des vies en facilitant des interventions médicales rapides et appropriées. Pour des implémentations futures, des ajustements supplémentaires, tels que l'optimisation des hyperparamètres et l'évaluation sur d'autres datasets, pourraient encore améliorer les performances des modèles.