IBN ZOHR UNIVERSITY FACULTY OF SCIENCES – AGADIR

Center of Excellence

Program: Computer Engineering and Embedded Systems

Machine Learning Project Report

Lab Group: 3

Prepared by: ZGAR LATIFA

OUBOURHAIL CHAYMA

Breast Cancer Classification Using Machine Learning Algorithms

Abstract

This project focuses on the use of machine learning algorithms to predict the nature of breast tumors based on clinical data. The dataset used is the Breast Cancer Wisconsin (Diagnostic) dataset. The goal is to develop and compare several classification models to identify the most effective one in distinguishing between benign and malignant tumors.

1. Introduction

Breast cancer is one of the leading causes of mortality among women. Accurate diagnosis is therefore essential to ensure timely and appropriate treatment. Machine learning can provide crucial support in predicting medical diagnoses. This project aims to evaluate several classification models applied to biopsy data from breast tumors.

2. Problem Statement

With the growing volume of medical data, traditional diagnostic methods are showing their limitations. How can we effectively predict whether a tumor is benign or malignant based on clinical measurements extracted from biopsies?

3. Dataset Description

- **Name:** Breast Cancer Wisconsin (Diagnostic)
- **Size:** 569 samples
- **Variables:** 30 features (statistical measurements of cell nuclei) with one target variable ('diagnosis')
- **Classes:** M (malignant), B (benign)

4. Models Used:

Logistic Regression

Logistic regression is a supervised classification algorithm used to predict binary categorical variables. In this report, we apply logistic regression using the SGDClassifier on the Breast Cancer Wisconsin dataset to predict whether a tumor is malignant (M) or benign (B).

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import SGDClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
# Chargement des données
dataset = pd.read_csv('Breast Cancer Wisconsin.csv')
```

The command is used to load data from a CSV file.

```
# Vérification des informations du dataset
dataset.info()
print(dataset.columns)
print(dataset.describe())
print(dataset.shape)
print(dataset.sample(5))
```

These commands allow for quick exploration of the dataset's structure and content. Each command displays the following information:

- Column data types
- Column names
- Descriptive statistics
- Dataset dimensions (rows and columns)
- 5 random
  sampl

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000
```

```python
# Suppression de la colonne inutile
if 'Unnamed: 32' in dataset.columns:
    dataset = dataset.drop('Unnamed: 32', axis=1)
```

Delete an empty column if it exists (such as "Unnamed: 32", which is often generated automatically).

```python
# Conversion de "M" et "B" en 0 et 1
dataset["diagnosis"] = dataset["diagnosis"].map({"M": 1, "B": 0})
```

And the following command is used to encode the target variable:

- M (malignant) → 1
- B (benign) → 0

```python
# Vérification des valeurs manquantes
print(dataset.isnull().sum())
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
```

The command displays the number of missing values per column.

```python
# Définition des variables
X = dataset.iloc[:, 3:].values  # Exclure la colonne "diagnosis"
y = dataset["diagnosis"].values
```

- Selects all columns starting from index 3
- values converts the data into a NumPy array
- y = dataset["diagnosis"].values retrieves the **diagnosis** column and converts it into a NumPy array

```python
# Normalisation des données
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

We normalize the data to improve the model's convergence. Each variable is standardized (mean of 0, standard deviation of 1).

```python
# Assurer que y est un vecteur
y = y.ravel()
```

We flatten y to obtain a one-dimensional vector, which is required for certain models.

```
# Séparer en train et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

We split the dataset into two parts:

- 80% to train the model (X_train, y_train)
- 20% to evaluate it (X_test, y_test)

```
# Initialisation et entraînement du modèle
model = SGDClassifier(max_iter=1000, loss='log_loss')
model.fit(X_train, y_train)
```

```
SGDClassifier
SGDClassifier(loss='log_loss')
```

We initialize a classifier using logistic regression (with the log_loss function), trained via stochastic gradient descent, and then fit it to the data.

```
# Prédictions et score
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.956140350877193
```

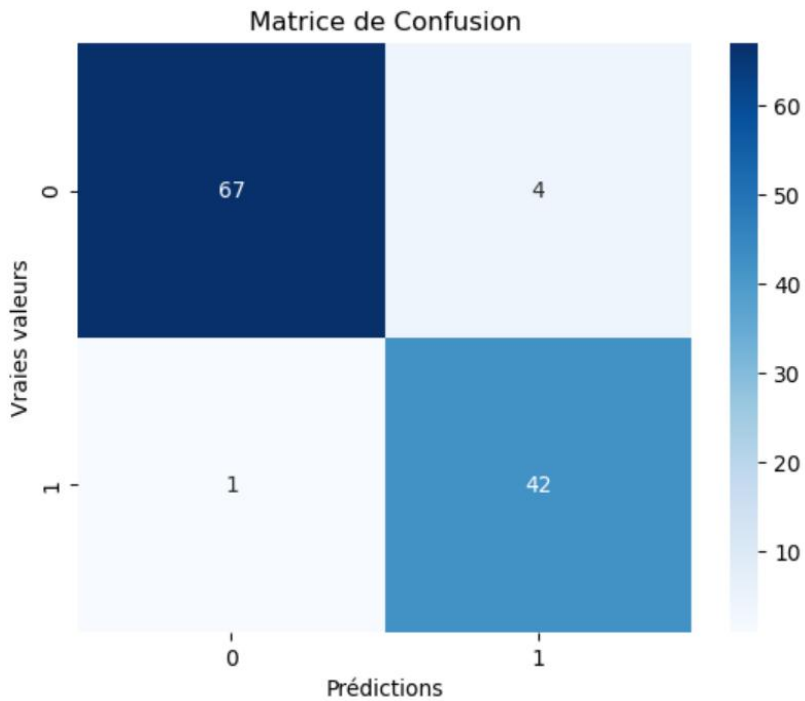
We use the .predict() method to generate predictions from the model trained on the test data. Next, we measure the model's accuracy, which is the proportion of correct predictions out of all predictions made.

Accuracy = 0.956140350877193

This means the model correctly predicted approximately **95.61%** of the cases in the test dataset.

```
# 1. Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()
```

We generate a confusion matrix to visualize the model's performance.




```
# Sélection uniquement des 2 premières caractéristiques pour l'affichage
X_vis = X_train[:, [26, 19]]  # Sélectionne uniquement les colonnes 26 (concave points_worst)et 19(radius_worst)
X_test_vis = X_test[:, [26, 19]]
```

We select two highly discriminative variables:

- concave points_worst (index 26)
- radius_worst (index 19)

This allows us to visualize the decision boundary in 2D.

```
# Entraînement du modèle avec seulement ces 2 caractéristiques
model_vis = SGDClassifier(max_iter=1000, loss='log_loss')
model_vis.fit(X_vis, y_train)
```

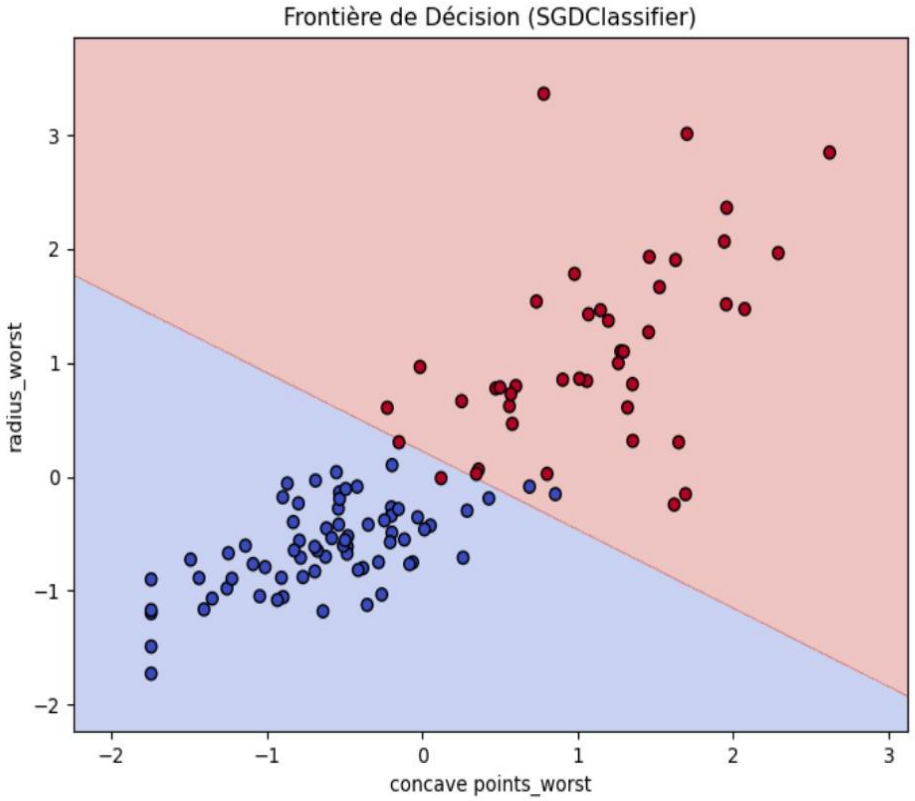
```
SGDClassifier
SGDClassifier(loss='log_loss')
```

We retrain a model using only these two variables.

```
def plot_decision_boundary(X, y, model, steps=1000, cmap='coolwarm'):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

    xx, yy = np.meshgrid(np.linspace(x_min, x_max, steps),
                         np.linspace(y_min, y_max, steps))
    grid = np.c_[xx.ravel(), yy.ravel()]
    preds = model.predict(grid).reshape(xx.shape)

    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, preds, alpha=0.3, cmap=cmap)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap, edgecolors='k')
    plt.xlabel('concave points_worst')
    plt.ylabel('radius_worst')
    plt.title('Frontière de Décision (SGDClassifier)')
    plt.show()

# ----------------------------
# Visualisation 2D sur 2 features
# ----------------------------
X_vis = X_train[:, [26, 19]]     # concave points_worst, radius_worst
X_test_vis = X_test[:, [26, 19]]

model_vis = SGDClassifier(max_iter=1000, loss='log_loss')
model_vis.fit(X_vis, y_train)

plot_decision_boundary(X_test_vis, y_test, model_vis)
```

As part of our study on tumor classification, we implemented a plot_decision_boundary() function to visualize how our model separates benign from malignant cases.

- Thousands of points spaced 0.01 units apart are generated to cover the entire plot.
- The model predicts the class (0 or 1) for each point in the grid.
- contourf(): Colors the regions based on the predicted class (blue for benign, orange for malignant).
- scatter(): Overlays the actual data points to compare with the predictions.
- The result shows that blue points (true benign cases) fall within the blue region, and orange points (true malignant cases) fall within the orange region.


Frontière de Décision (SGDClassifier)
radius_worst
concave points_worst
3
2
1
0
−1
−2
−2
−1
0
1
2
3

K-Nearest Neighbors (KNN) The KNN

algorithm is a classification model used to predict whether a tumor is benign or malignant. The principle is simple: a new tumor is compared to the K closest tumors in the dataset (based on their features). Then, the model checks the majority class among these neighbors: if most are malignant, it predicts “malignant”; otherwise, it predicts “benign.” ».

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
```

- We import **NumPy** for mathematical operations.
- **matplotlib.pyplot** and **seaborn** are used to display visualizations (such as the confusion matrix or error plots).
- **pandas** is used to read and manipulate the dataset.
- We use **StandardScaler** and **MinMaxScaler** to normalize the data.
- **train_test_split** is used to divide the data into training and testing sets.
- **accuracy_score**, **confusion_matrix**, and **classification_report** are used to evaluate the model’s performance.
- Finally, **KNeighborsClassifier** is the KNN model we will train.

```
# Chargement des données
dataset = pd.read_csv('Breast Cancer Wisconsin.csv')
```

We load the CSV file containing breast cancer data into a DataFrame named dataset.

```python
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
```

```python
print(dataset.columns)
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

```python
print(dataset.describe())
```

```
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  ...  texture_worst  perimeter_worst   area_worst  \
count     569.000000  ...     569.000000       569.000000   569.000000
mean        0.181162  ...      25.677223       107.261213   880.583128
std         0.027414  ...       6.146258        33.602542   569.356993
min         0.106000  ...      12.020000        50.410000   185.200000
25%         0.161900  ...      21.080000        84.110000   515.300000
50%         0.179200  ...      25.410000        97.660000   686.500000
```

```python
print(dataset.shape)
```

```
(569, 33)
```

```python
print(dataset.sample(5))
```

```
            id diagnosis  radius_mean  texture_mean  perimeter_mean  \
150  871001501         B        13.00         20.78           83.51
541     921386         B        14.47         24.99           95.81
221    8812818         B        13.56         13.90           88.59
94      862028         M        15.06         19.83          100.30
404     904969         B        12.34         14.95           78.29

     area_mean  smoothness_mean  compactness_mean  concavity_mean  \
150      519.4          0.11350           0.07589         0.03136
541      656.4          0.08837           0.12300         0.10090
221      561.3          0.10510           0.11920         0.07860
94       705.6          0.10390           0.15530         0.17000
404      469.1          0.08682           0.04571         0.02109

     concave points_mean  ...  texture_worst  perimeter_worst  area_worst  \
150              0.02645  ...          24.11            90.82       616.7
541              0.03890  ...          31.73           113.50       808.9
221              0.04451  ...          17.13           101.10       686.6
94               0.08815  ...          24.23           123.50      1025.0
404              0.02054  ...          16.85            84.11       533.1

     smoothness_worst  compactness_worst  concavity_worst  \
150            0.1297            0.11050          0.08112
541            0.1340            0.42020          0.40400
221            0.1376            0.26980          0.25770
94             0.1551            0.42030          0.52030
404            0.1048            0.06744          0.04921

     concave points_worst  symmetry_worst  fractal_dimension_worst  \
150               0.06296          0.3196                  0.06435
541               0.12050          0.3187                  0.10230
```

```python
# Suppression de la colonne inutile
if 'Unnamed: 32' in dataset.columns:
    dataset = dataset.drop('Unnamed: 32', axis=1)
```

We remove the column "Unnamed: 32" if it exists, as it is an empty and unnecessary column.

```python
# Conversion de "M" et "B" en 0 et 1
dataset["diagnosis"] = dataset["diagnosis"].map({"M": 1, "B": 0})
```

We replace "M" (malignant) with 1 and "B" (benign) with 0 to convert the classes into numeric format, which is required for the algorithm.

```
# Vérification des valeurs manquantes
print(dataset.isnull().sum())
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
```

We check that there are no missing values in the dataset. The output confirms that all columns contain 0 missing values.

```
# Définition des variables
X = dataset.iloc[:, 3:].values  # Exclure la colonne "diagnosis"
y = dataset["diagnosis"].values
```

- We select the features starting from column 3 to the end (excluding ID and name).
- y contains the target variable (0 or 1).

```
# Assurer que y est un vecteur
y = y.ravel()
```

ravel() transforms y into a 1D array, which is necessary for training.

```
# Séparer en train et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- We split the data into 80% for training and 20% for testing.
- Setting random_state=42 ensures reproducible results.

```
# Normalisation des données
scaler = StandardScaler()
scaler.fit(X_train)
X_train= scaler.transform(X_train)
X_test= scaler.transform(X_test)
```

Before normalizing the data with StandardScaler, you must first train the scaler by calling .fit() on the training data. Then, you can apply .transform() to normalize the values. This step is important to avoid the NotFittedError.

```
classifier=KNeighborsClassifier(n_neighbors=38)
```

```
classifier.fit(X_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=38)
```

We create a KNN model with 38 neighbors.

We train the model using the training data.

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

```
print (accuracy_score(y_test,y_pred)*100)
```

```
96.49122807017544
```

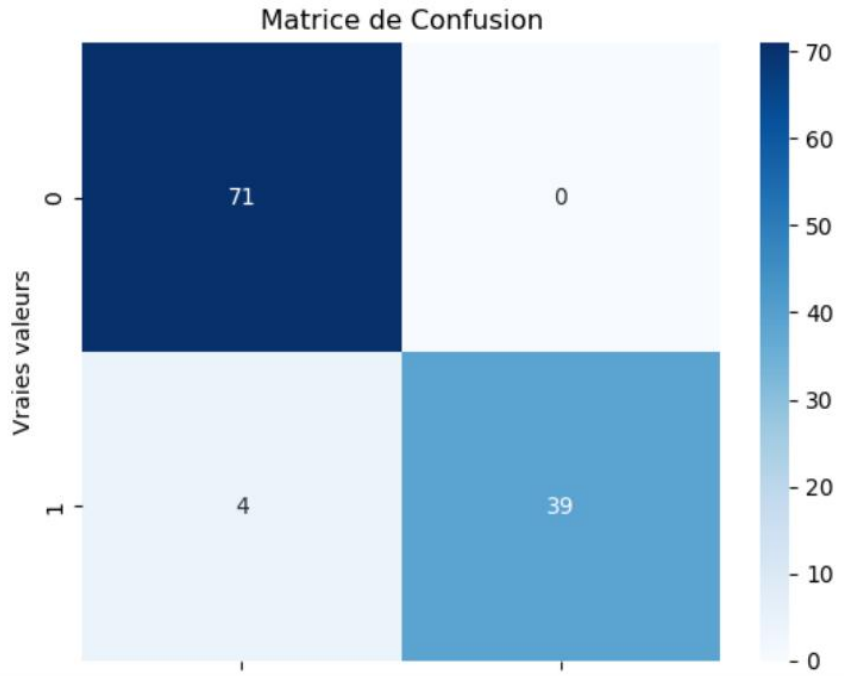
- In the first line, we use our KNN model, which has already been trained on the X_train dataset using classifier.fit, to make predictions on the test set X_test.
- With classifier.predict, we ask the model to guess which class each observation in X_test belongs to — malignant or benign cancer.
- The result is a vector y_pred containing the model's predictions for each sample.

In the second line, we evaluate how accurate the model's predictions are by comparing:

✓ y_test: the true classes of the test set,
✓ y_pred: the predicted classes from the model.

The output of the command, **96.49%**, means that the model correctly predicted the diagnosis for approximately **96 out of 100 patients**.

```
# 1. Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()
```



Interpretation of Results

1. **True Negatives (TN = 71):**

- ✧ 71 benign tumors were correctly identified as benign.
- ✧ These are cases where the model predicted “healthy” and the patient was indeed healthy.

2. **False Positives (FP = 0):**

- ✧ No cases where a benign tumor was incorrectly classified as malignant.
- ✧ This is very important, as avoiding false positives helps prevent unnecessary treatments and reduces patient stress.

3. **False Negatives (FN = 4):**

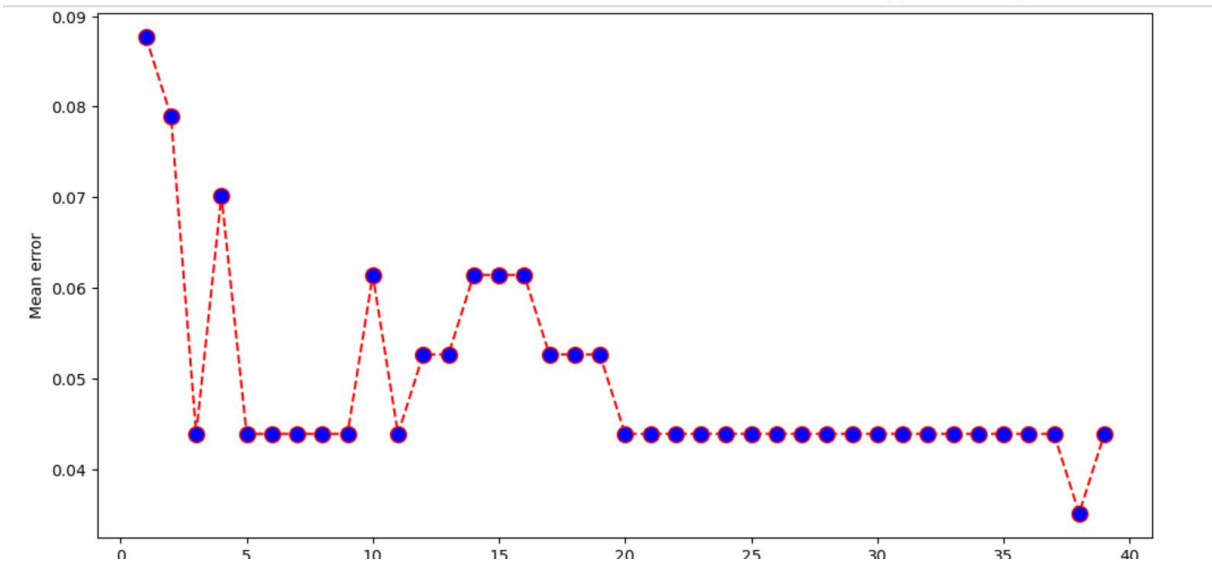
- ✧ 4 malignant tumors were incorrectly classified as benign.
- ✧ These are the most dangerous cases in oncology, as they create a false sense of security and could delay necessary treatment.

4. **True Positives (TP = 39):**

- ✧ 39 malignant tumors were correctly detected.

```
error =[]
for i in range(1,40):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i=knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12,6))
plt.plot(range(1,40),error,color='red',linestyle='dashed',marker='o',markerfacecolor='blue',markersize=10)
plt.title('Error Rate K value')
plt.xlabel('K value')
plt.ylabel('Mean error')
plt.show()
```

This for loop is used to test different values of K between 1 and 39 to determine which one performs best.



Relationship Between Error Rate and K Value

The curve shows a relationship between the error rate and the value of K:

✧ **High error rate** for small values of K (K = 1 to 20)
✧ **Minimum error** observed around K > 20

Therefore, to choose a reliable K value, it is recommended to select it within the range of **35 to 40**.

Breast Cancer Classification Using Decision Tree

In our project, a decision tree is a supervised learning model used to automatically classify tumors. It considers various features such as cell size, texture, and concavity.

Thanks to its simple tree-like structure — with nodes, branches, and leaves — it enables fast and interpretable decisions.

This is especially useful for understanding the key criteria that influence the diagnosis.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
```

We imported the DecisionTreeClassifier library.

Loading and exploring the dataset:

```
# Chargement des données
dataset = pd.read_csv('Breast Cancer Wisconsin.csv')

# Vérification des informations du dataset
dataset.info()
print(dataset.columns)
print(dataset.describe())
print(dataset.shape)
print(dataset.sample(5))

# Suppression de la colonne inutile
if 'Unnamed: 32' in dataset.columns:
    dataset = dataset.drop('Unnamed: 32', axis=1)

# Conversion de "M" et "B" en 0 et 1
dataset["diagnosis"] = dataset["diagnosis"].map({"M": 1, "B": 0})

# Vérification des valeurs manquantes
print(dataset.isnull().sum())
```

Variable definition:

```
# Définition des variables
X = dataset.iloc[:, 3:].values  # Exclure la colonne "diagnosis"
y = dataset["diagnosis"].values
y = y.ravel()
```

Splitting the dataset into training and test data, and normalizing the data:

```
# Séparer en train et test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalisation des données
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

We created a decision tree model with a maximum depth of 6.

It was trained (fit) using the normalized training data.

```
# Création et entraînement du modèle Decision Tree
classifier = DecisionTreeClassifier(max_depth=6, random_state=42)
classifier.fit(X_train, y_train)
```

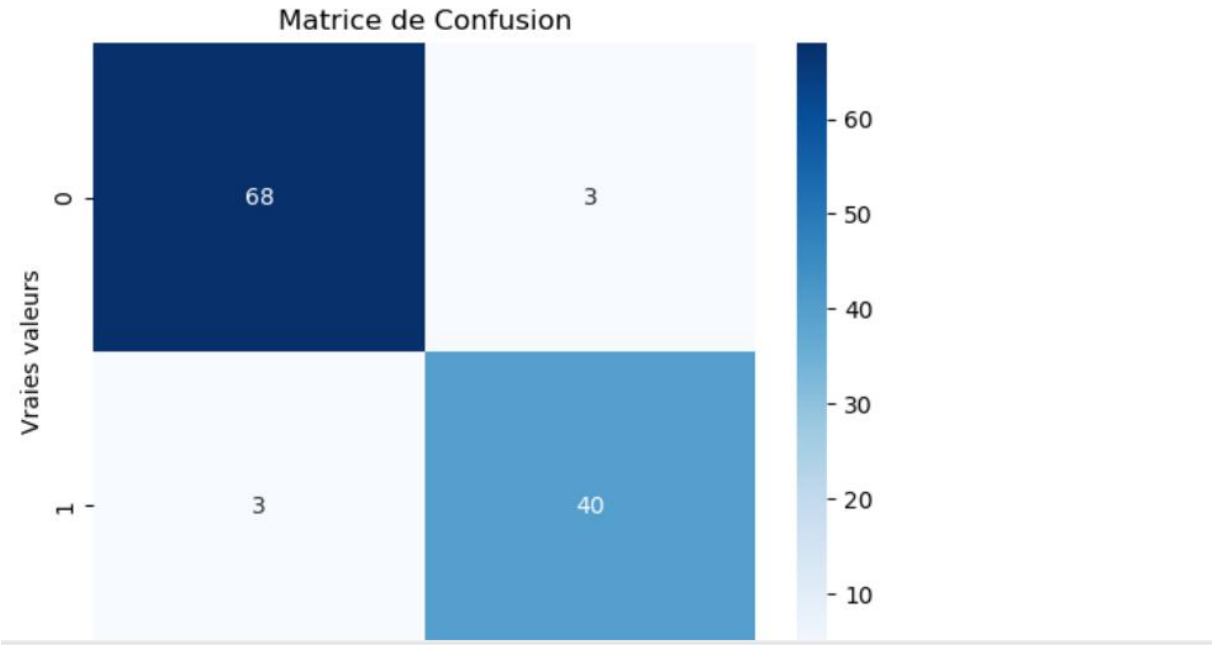
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, random_state=42)
```

Prediction on the test data:

```
# Prédictions
y_pred = classifier.predict(X_test)
print(f"Précision du modèle: {accuracy_score(y_test, y_pred) * 100:.2f}%")
```

Confusion matrix:

```python
# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()
```



Confusion Matrix Explanation

Clinical Interpretation

- **True Negatives (68):**

  68 patients were healthy, and the model correctly identified them as such.

  Example: A biopsy confirms the absence of cancer, and the model agrees.

- **False Positives (3):**

  3 patients were healthy, but the model falsely suspected cancer.
  Impact:

  Unnecessary follow-up exams (MRI, biopsy)

  Increased stress for the patient

- **False Negatives (3):**

  3 patients had cancer, but the model failed to detect it.

  Serious issue:

  Delayed treatment

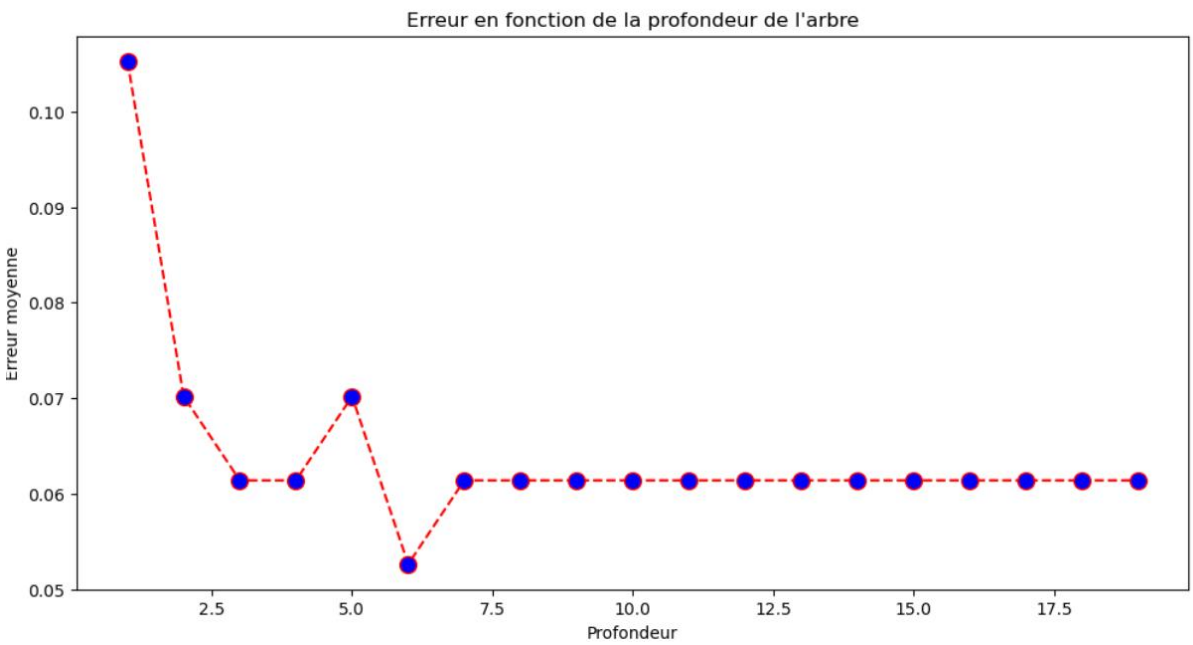
  Risk of disease progression

- **True Positives (40):**

40 patients had cancer, and the model correctly detected it. **Good news:** Early diagnosis → timely treatment

```
# Test de performance avec différentes profondeurs de l'arbre
errors = []
for i in range(1, 20):
    dt = DecisionTreeClassifier(max_depth=i, random_state=42)
    dt.fit(X_train, y_train)
    pred_i = dt.predict(X_test)
    errors.append(np.mean(pred_i != y_test))
```

```
# Graphique des erreurs en fonction de la profondeur
plt.figure(figsize=(12, 6))
plt.plot(range(1, 20), errors, color='red', linestyle='dashed', marker='o', markerfacecolor='blue', markersize=10)
plt.title('Erreur en fonction de la profondeur de l\'arbre')
plt.xlabel('Profondeur')
plt.ylabel('Erreur moyenne')
plt.show()
```



Conclusion:

The use of machine learning algorithms to predict the nature of breast tumors based on clinical data—as demonstrated through the analysis of the Breast Cancer Wisconsin (Diagnostic) dataset—highlights the effectiveness of these models in supporting medical diagnosis.

Among the models tested—**logistic regression**, **K-nearest neighbors (KNN)**, and **decision tree**—**KNN achieved the highest accuracy**, with a score of **96.49%**, suggesting a superior ability to distinguish between benign and malignant tumors.

These results underscore the potential of machine learning to enhance diagnostic tools. However, it remains essential to continue improving these models, particularly in terms of **robustness** and **generalization**, to ensure their applicability in real-world clinical settings.