Breast Cancer Prediction Model:

# Breast Cancer Prediction Model Using Machine Learning and Neural Network

NAME: DANISH SHAIKH

ENROLLMENT NO: A70405222137

CLASS: DATA SCIENCE

SEM: 6

## 1. Introduction

Breast cancer prediction involves identifying whether a tumor is malignant or benign based on features extracted from medical data. This task is crucial for early diagnosis, treatment planning, and improving patient outcomes. With advancements in machine learning, particularly neural networks, automated classification of breast cancer has become more accurate. This project aims to predict breast cancer using the scikit-learn breast cancer dataset, leveraging machine learning techniques to achieve high accuracy in real-world applications.

## 2. Objective

- To build an automated system capable of classifying breast cancer cases as malignant or benign.

- To train the model using the scikit-learn breast cancer dataset.

- To evaluate the model's performance on a validation set.

- To deploy the model for real-time breast cancer prediction.

## 3. Methodology

a. Dataset Collection

- Use the scikit-learn breast cancer dataset, which contains 569 samples with 30 features (e.g., radius, texture, perimeter) and two classes (malignant and benign).

- The dataset is publicly available and well-suited for binary classification tasks.

b. Data Preprocessing

- Normalize feature values to a range of 0 to 1 using StandardScaler or MinMaxScaler.

- Handle missing values, if any, though the scikit-learn dataset is clean.

- Split the dataset into features (X) and labels (y).

c. Model Building (Neural Network)

- Use a Multi-Layer Perceptron (MLP) neural network architecture with:

- Input layer matching the number of features (30).

- Hidden layers with ReLU activation to capture complex patterns.

- Output layer with sigmoid activation for binary classification.

- Alternatively, explore other classifiers like Logistic Regression or Random Forest for comparison.

d. Training and Evaluation

- Split the dataset into training (70%), validation (15%), and test (15%) sets.

- Train the neural network using an optimizer like Adam and a loss function like Binary Cross entropy.

- Evaluate the model using accuracy, precision, recall, and F1-score as metrics.

e. Testing and Prediction

- Test the model on the test set to assess its performance on unseen data.

- Deploy the model to predict breast cancer outcomes in real-time.

## 4. Tools and Libraries

| Tool/Library | Purpose

|--------------------- |----------------------------------------------|

| Python | Programming language

| scikit-learn | Dataset access, preprocessing, and metrics

| Pytorch | Deep learning framework for neural network

| NumPy | Numerical operations and array manipulation

| Pandas | Data manipulation and analysis

| Matplotlib / Seaborn | Visualization of performance metrics

## 5. Expected Outcome

- The model should classify breast cancer cases into malignant or benign categories.

- The accuracy of the model is expected to range between 90% and 98%, depending on the neural network architecture and hyperparameter tuning.

- A working breast cancer prediction system capable of real-time predictions.

## 6. Applications

- Early Diagnosis: Assisting medical professionals in identifying breast cancer at an early stage.

- Medical Decision Support: Providing automated predictions to support doctors in treatment planning.

- Healthcare Apps: Developing applications for patients to assess risk based on medical data.

- Research: Supporting studies on breast cancer by analyzing patterns in medical features.

## 7. Future Scope

- Transfer Learning: Adapting pre-trained neural networks for improved performance on smaller datasets.

- Feature Engineering: Incorporating additional features like genetic markers or imaging data.

- Real-time Integration: Deploying the model in hospital systems for real-time diagnosis.

- Multi-class Classification: Extending the model to predict cancer subtypes or stages.

# Code:

*# Import necessary libraries*

import torch

import torch.nn as nn

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, precision\_score, r2\_score, f1\_score

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

*# 0 - Load and prepare the dataset*

bc = datasets.load\_breast\_cancer()

X, y = bc.data, bc.target

n\_samples, n\_features = X.shape

*# Split the dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Standardize features*

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

*# Convert NumPy arrays to PyTorch tensors*

X\_train = torch.from\_numpy(X\_train.astype(np.float32))

X\_test = torch.from\_numpy(X\_test.astype(np.float32))

y\_train = torch.from\_numpy(y\_train.astype(np.float32)).view(-1, 1)

y\_test = torch.from\_numpy(y\_test.astype(np.float32)).view(-1, 1)

*# 1 - Define the logistic regression model*

class LogisticRegression(nn.Module):

    def \_\_init\_\_(self, n\_input\_features):

        super(LogisticRegression, *self*).\_\_init\_\_()

*self*.layer1 = nn.Linear(n\_input\_features, 1)

    def forward(self, x):

        return torch.sigmoid(*self*.layer1(x))

*# Initialize model, loss, and optimizer*

model = LogisticRegression(n\_features)

loss\_fn = nn.BCELoss()

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

*# 2 - Training loop*

epochs = 100

loss\_list = []

for epoch in range(epochs):

    y\_pred = model(X\_train)

    loss = loss\_fn(y\_pred, y\_train)

    loss.backward()

    optimizer.step()

    optimizer.zero\_grad()

    loss\_list.append(loss.item())

    if epoch % 10 == 0:

        print(f"Epoch {epoch+1}: Loss = {loss.item():.4f}")

*# 3 - Plot the loss over epochs*

plt.figure(figsize=(8, 4))

plt.plot(range(epochs), loss\_list, label='Training Loss')

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.title("Loss Curve")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

*# 4 - Evaluation*

with torch.no\_grad():

    y\_predicted = model(X\_test)

    y\_predicted\_cls = y\_predicted.round()

*# Convert tensors to NumPy arrays*

    y\_pred\_numpy = y\_predicted\_cls.numpy().astype(int).flatten()

    y\_test\_numpy = y\_test.numpy().astype(int).flatten()

*# Accuracy*

    acc = y\_predicted\_cls.eq(y\_test).sum() / float(y\_test.shape[0])

    print(f'Accuracy: {acc:.4f}')

*# Precision, R2, F1 Score*

    precision = precision\_score(y\_test\_numpy, y\_pred\_numpy)

    r2 = r2\_score(y\_test\_numpy, y\_pred\_numpy)

    f1 = f1\_score(y\_test\_numpy, y\_pred\_numpy)

    print(f"Precision Score: {precision:.4f}")

    print(f"R2 Score: {r2:.4f}")

    print(f"F1 Score: {f1:.4f}")

*# 5 - Confusion Matrix*

cm = confusion\_matrix(y\_test\_numpy, y\_pred\_numpy)

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

            xticklabels=['Benign', 'Malignant'],

            yticklabels=['Benign', 'Malignant'])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.tight\_layout()

plt.show()

# Result & Graph:

A computer screen with a black background

AI-generated content may be incorrect.



