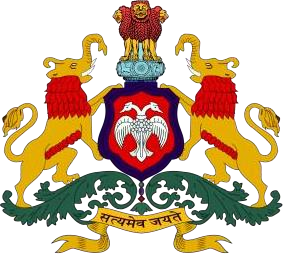
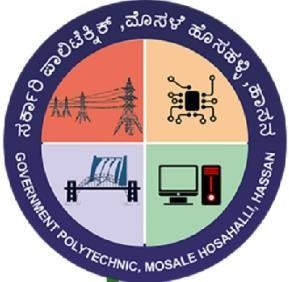
Government of Karnataka

Department of Technical Education

Bangalore – 560001

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOVT. POLYTECHNIC

MOSALE HOSAHALLI-573212

**2025-2026**

**SPECIALIZATION PATHWAY**

**ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

**SUBMITTED BY:**

**NAME: LAKSHMI**

**REG NO: 189CS23026**

**SUBMITTED TO:**

**Praveen Kumar K**

**Selection Grade Lecturer**

**Dept. of CSE**

**Table Content**

**SPECIALIZATION PATHWAY**

**ARTIFICIAL INTELLIGENCE & MACHINE LEARING**

**Problem 1:**

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Content** | **Page no** |
| **1** | **Introduction to Regression** | **3** |
| **2** | **Implementation Program of ML&DL** | **4-9** |
| **3** | **Introduction to Classification** | **10** |
| **4** | **Implementation Program of ML&DL** | **11-16** |
| **5** | **Analyse the performance of ML and DL** | **17-19** |

**Problem 2:**

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Content** | **Page no** |
| **1** | **Problem Statement** | **20** |
| **2** | **Project plan, Product Backlog** | **21** |

**Problem 3:**

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Content** | **Page No** |
| **1** | **GitHub & Repository** | **22** |

**Regression-Rebuild with Deep Learning model**

**1.Introduction**

**Introduction to Regression**

Regression is a core concept in machine learning (ML) and statistics that focuses on predicting a continuous numerical value based on input data. Unlike classification (which predicts categories), regression deals with outputs that are quantitative and can take on any real value.

**- Machine Learning (ML):** Uses algorithms that learn from data without being explicitly programmed.

**- Deep Learning (DL):** A subset of ML that uses Artificial Neural Networks (ANNs) with multiple hidden layers.

**In this project:**

**- Regression Task →** Diabetes Prediction

**- Classification Task →** Heart Disease Prediction

**2. Regression – Diabetes Prediction**

**🔹 Machine Learning (ML) Approach**

**Common Algorithm**: Linear Regression, which assumes a straight-line relationship between features and the targe**t.**

**Advantages:**

Simple and easy to interpret.

Works well on small datasets.

**Limitations:**

Accuracy reduces as dataset size and complexity increase.

Cannot capture complex non-linear relationships.

**Diabetes Prediction in ML Program:**

# diabetes\_ml\_tf.py

import pandas as pd

import tensorflow as tf

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load dataset

data = pd.read\_csv(“diabetes.csv")

print(data.head())

X = data.drop("Outcome", axis=1)

y = data["Outcome"]

# Scale data

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split data

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

# ML Model (like Linear Regression)

model = tf.keras.Sequential([

tf.keras.layers.Dense(1, input\_shape=(X.shape[1],)) # Single neuron, no activation])

model.compile(optimizer="adam", loss="mse")

# Train

history = model.fit(X\_train, y\_train, epochs=100, verbose=0)

# Predict

y\_pred = model.predict(X\_test).flatten()

# Evaluation

print("ML (TF) Regression Results:")

print("MSE:", mean\_squared\_error(y\_test, y\_pred))

print("R2 Score:", r2\_score(y\_test, y\_pred))

import matplotlib.pyplot as plt

import numpy as np

# Plot training loss curve

plt.plot(history.history['loss'])

plt.title('Training Loss (MSE)')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.show()

# Plot Actual vs Predicted

plt.scatter(y\_test, y\_pred, alpha=0.7)

plt.xlabel("Actual Values")

plt.ylabel("Predicted Values")

plt.title("Actual vs Predicted (Regression)")

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red') # ideal line

plt.show()

**Output:**

Pregnancies Glucose BloodPressure ... DiabetesPedigreeFunction Age Outcome

0 6 148 72 ... 0.627 50 1

1 1 85 66 ... 0.351 31 0

2 8 183 64 ... 0.672 32 1

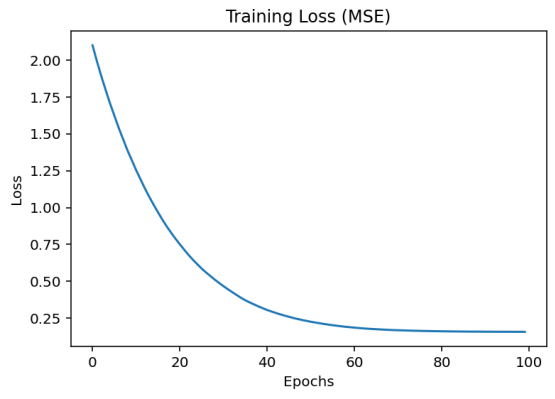
3 1 89 66 ... 0.167 21 0

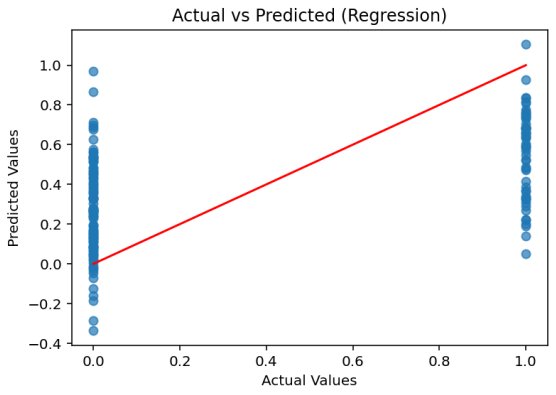
4 0 137 40 ... 2.288 33 1

ML (TF) Regression Results:

MSE: 0.1709032654762268

R2 Score: 0.2556213140487671





**🔹 Deep Learning (DL) Approach**

**Structure**: Input layer (features) → hidden layers (non-linear transformations) → output layer (predicted diabetes measure).

**Advantages:**

Learns complex patterns and non-linear relationships.

More accurate when dataset is large.

**Limitations:**

Requires more data and computational resources.

Less interpretable compared to ML.

**Diabetes Prediction in DL Program:**

# diabetes\_dl\_tf.py

import pandas as pd

import tensorflow as tf

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load dataset

data = pd.read\_csv("diabetes.csv")

print(data.head()) # Corrected line

X = data.drop("Outcome", axis=1)

y = data["Outcome"]

# Scale data

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split data

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

# DL Model

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation="relu", input\_shape=(X.shape[1],)),

tf.keras.layers.Dense(32, activation="relu")

tf.keras.layers.Dense(1) # Regression output])

model.compile(optimizer="adam", loss="mse")

# Train

history = model.fit(X\_train, y\_train, epochs=100, verbose=0)

# Predict

y\_pred = model.predict(X\_test).flatten()

# Evaluation

print("DL Regression Results:")

print("MSE:", mean\_squared\_error(y\_test, y\_pred))

print("R2 Score:", r2\_score(y\_test, y\_pred))

import matplotlib.pyplot as plt

import numpy as np

# Plot training loss curve

plt.plot(history.history['loss'])

plt.title('Training Loss (MSE)')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.show()

# Plot Actual vs Predicted

plt.scatter(y\_test, y\_pred, alpha=0.7)

plt.xlabel("Actual Values")

plt.ylabel("Predicted Values")

plt.title("Actual vs Predicted (Regression)")

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red') # ideal line

plt.show()

**Output:**

Pregnancies Glucose BloodPressure ... DiabetesPedigreeFunction Age Outcome

0 6 148 72 ... 0.627 50 1

1 1 85 66 ... 0.351 31 0

2 8 183 64 ... 0.672 32 1

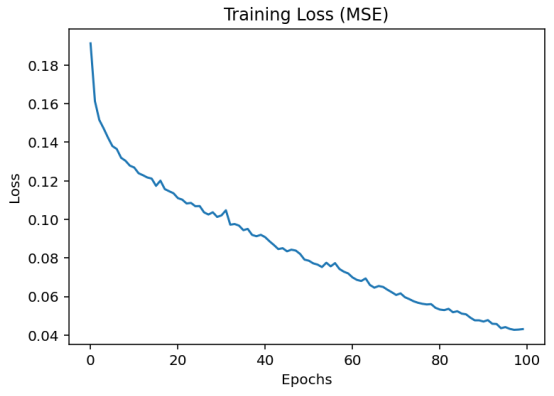
3 1 89 66 ... 0.167 21 0

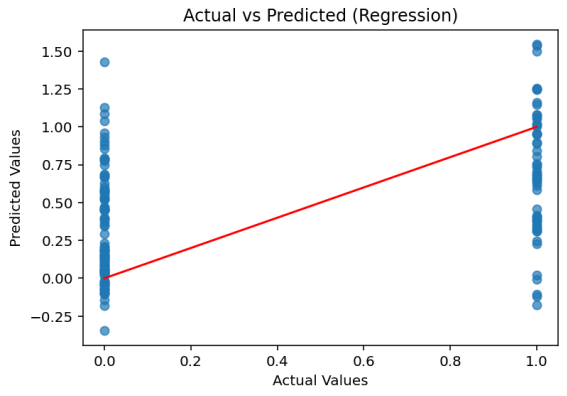
4 0 137 40 ... 2.288 33 1

DL Regression Results:

MSE: 0.23747684061527252

R2 Score: -0.034343600273132324





**Classification -Rebuild with Machine Learning model**

**Introduction to Classification**

Classification is a fundamental concept in machine learning (ML) and data science where the goal is to predict the category (class/label) of given input data based on past observations. It is a type of supervised learning, meaning the model is trained on a labelled dataset that already contains input-output pairs.

**Types of Classification**:

1. Binary Classification – Only two possible classes (e.g., Pass/Fail).

2. Multi-class Classification – More than two classes (e.g., classifying fruits into Apple, Mango, Orange).

3. Multi-label Classification – A single instance can belong to multiple classes simultaneously (e.g., tagging a movie as Action, Comedy, and Romance).

4. Imbalanced Classification

Definition: One class has much fewer samples compared to the other(s).

**3. Classification – Heart Disease Prediction**

**🔹 Machine Learning (ML) Approach**

**Concept:** Classification in ML predicts categorical values (e.g., “heart disease present” or “not present”).

**Example:** Predicting whether a patient has heart disease based on features like age, cholesterol, and blood pressure.

**Common Algorithm:** Logistic Regression, which uses probabilities to classify data into two categories.

**Advantages:**

Easy to train and interpret.

Suitable for small to medium datasets.

**Limitations:**

May not capture complex interactions among features.

Accuracy limited on high-dimensional data.

**Heart Disease in ML Program:**

# heart\_ml\_tf.py

import pandas as pd

import tensorflow as tf

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report

# Load dataset

data = pd.read\_csv("C:/Users/Punith H J/OneDrive/Documents/heart.csv")

X = data.drop("target", axis=1)

y = data["target"]

# Scale

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split

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

# ML Model (Logistic Regression = single dense + sigmoid)

model = tf.keras.Sequential([

tf.keras.layers.Dense(1, activation="sigmoid", input\_shape=(X.shape[1],))])

model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])

# Train

history = model.fit(X\_train, y\_train, epochs=100, verbose=0)

# Predict

y\_pred = (model.predict(X\_test) > 0.5).astype("int32")

# Evaluation

print("ML (TF) Classification Results:")

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

# --- Alternative plot for classification ---

import matplotlib.pyplot as plt

# Extract history

loss = history.history['loss']

accuracy = history.history.get('accuracy', None)

epochs = range(1, len(loss)+1)

# Plot Loss

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

plt.plot(epochs, loss, 'r-', marker='o', label='Loss')

plt.title('Training Loss over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.grid(True)

plt.legend()

plt.show()

# Plot Accuracy (if available)

if accuracy:

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

plt.plot(epochs, accuracy, 'b-', marker='s', label='Accuracy')

plt.title('Training Accuracy over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.grid(True)

plt.legend()

plt.show()

**Output:**

Accuracy: 0.8524590163934426

ML (TF) Classification Results:

Accuracy: 0.8524590163934426

precision recall f1-score support

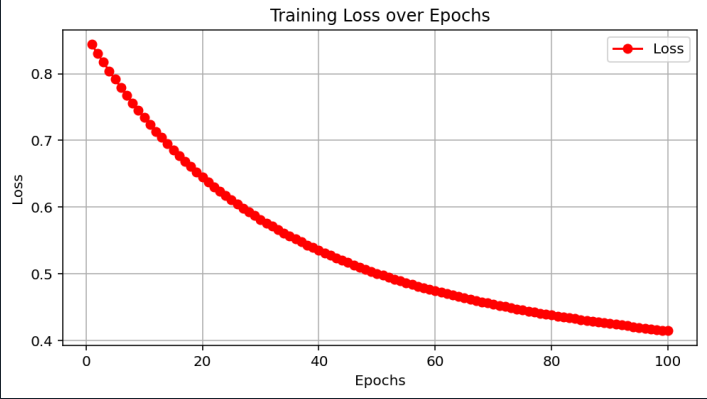
0 0.86 0.83 0.84 29

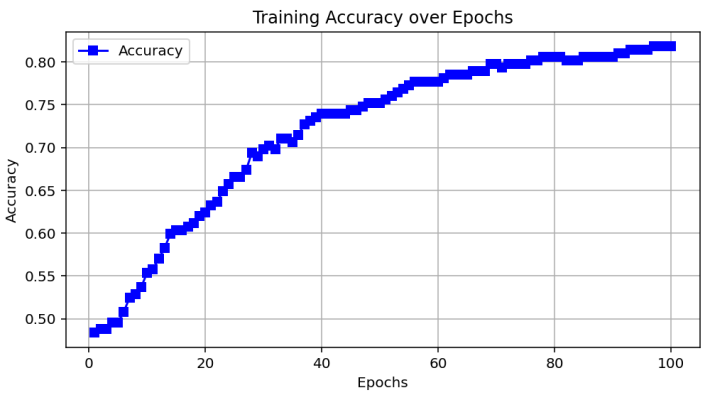
1 0.85 0.88 0.86 32

accuracy 0.85 61

macro avg 0.85 0.85 0.85 61

weighted avg 0.85 0.85 0.85 61





**🔹 Deep Learning (DL) Approach**

Classification is the task of assigning input data into predefined categories or classes.

In Deep Learning, classification is typically performed using artificial neural networks (ANNs), especially deep neural networks (DNNs), convolutional neural networks (CNNs), or recurrent neural networks (RNNs) depending on the data type.

The network automatically extracts features from raw data (images, text, audio, or tabular) without requiring manual feature engineering, which is a major advantage over traditional ML.

**Structure:** Input layer → hidden layers → output layer with sigmoid (binary) or softmax (multi-class) activation.

**Advantages:**

Captures hidden patterns and non-linear relationships.

Improves accuracy when dataset is large.

**Limitations:**

Requires more training time and computing power.

**Heart Disease in DL Program:**

# heart\_dl\_tf.py

import pandas as pd

import tensorflow as tf

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report

# Load dataset

data = pd.read\_csv("C:/Users/Punith H J/OneDrive/Documents/heart.csv")

print(data.head())

X = data.drop("target", axis=1)

y = data["target"]

# Scale

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split

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

# DL Model

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation="relu", input\_shape=(X.shape[1],)),

tf.keras.layers.Dense(32, activation="relu"),

tf.keras.layers.Dense(1, activation="sigmoid") # Binary classification])

model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])

# Train

history = model.fit(X\_train, y\_train, epochs=100, verbose=0)

# Predict

y\_pred = (model.predict(X\_test) > 0.5).astype("int32")

# Evaluation

print("DL Classification Results:")

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

# --- Alternative plot for classification ---

import matplotlib.pyplot as plt

# Extract history

loss = history.history['loss']

accuracy = history.history.get('accuracy', None)

epochs = range(1, len(loss)+1)

# Plot Loss

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

plt.plot(epochs, loss, 'r-', marker='o', label='Loss')

plt.title('Training Loss over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.grid(True)

plt.legend()

plt.show()

if accuracy:

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

plt.plot(epochs, accuracy, 'b-', marker='s', label='Accuracy')

plt.title('Training Accuracy over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.grid(True)

plt.legend()

plt.show()

**Output:**

Accuracy: 0.8524590163934426

DL Classification Results:

precision recall f1-score support

0 0.86 0.83 0.84 29

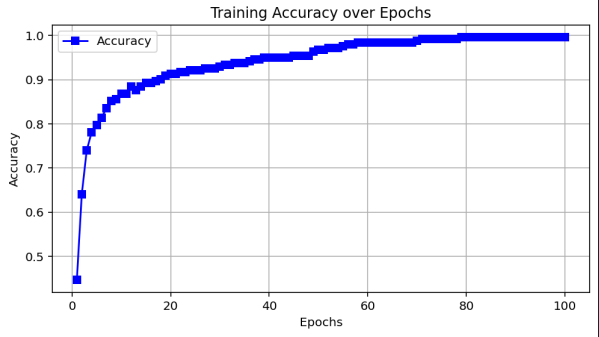
1 0.85 0.88 0.86 32

accuracy 0.85 61

macro avg 0.85 0.85 0.85 61

weighted avg 0.85 0.85 0.85 61



****

**Analyze the performance of ML and DL**

**🔹 1. Data Dependency**

**Machine Learning (ML):**

Performs well even on small to medium datasets (hundreds or thousands of samples).

Example: Logistic Regression or Decision Trees can classify heart disease with limited data.

Limitation: Accuracy may drop if the dataset grows very complex or has high-dimensional features.

**Deep Learning (DL):**

Needs large datasets to achieve high performance.

Example: ANN for diabetes regression needs more data to properly adjust weights.

Advantage: Learns directly from raw data without much manual preprocessing.

**🔹 2. Training Time & Resources**

**ML:**

Training is fast and requires low computational power.

Can be executed on CPUs.

Suitable for classroom projects, smaller datasets, or real-time predictions with limited hardware.

**DL:**

Training is slow and computationally heavy due to millions of parameters.

Requires GPUs/TPUs for faster convergence.

Higher memory and storage requirements.

**🔹 3. Feature Engineering**

**ML:**

Relies on manual feature selection and engineering.

Domain expertise is crucial (e.g., selecting blood pressure, age, BMI for diabetes).

Simpler models cannot automatically extract hidden patterns.

**DL:**

Performs automatic feature extraction.

Neural networks adjust weights to identify complex relationships between inputs and outputs.

Advantageous when raw data (like images, ECG signals, or lab reports) is directly used.

🔹 **4. Accuracy & Performance**

**ML:**

Provides baseline performance and decent accuracy on smaller datasets.

Example: Linear Regression (diabetes) and Logistic Regression (heart disease) give interpretable results.

Accuracy may plateau as problem complexity grows.

**DL:**

higher accuracy by modeling complex non-linear patterns.

Example: ANN regression can reduce error compared to Linear Regression, and ANN classification can outperform Logistic Regression.

Improves generalization if dataset is large enough.

**🔹 5. Interpretability**

**ML:**

More transparent and interpretable.

Example: Coefficients in Linear/Logistic Regression show which medical factors impact diabetes/heart disease most.

Useful for healthcare, where explainability is essential.

**DL:**

Acts like a black box – difficult to interpret why a decision was made.

Example: ANN may give higher accuracy but cannot clearly explain which exact feature caused the prediction.

This is a drawback in sensitive domains like medicine.

**🔹 6. Scalability & Flexibility**

**ML:**

Good for structured/tabular data.

Struggles with large unstructured data (images, ECG signals, text).

**DL:**

Highly scalable and flexible.

Works extremely well with unstructured and high-dimensional data.

Example: CNNs for medical images or ECG signals outperform ML models.

**🔹 7. Generalization**

**ML:**

Better on small datasets; less prone to overfitting if regularized properly.

May underfit on large complex data.

**DL:**

Can generalize well on large datasets but may overfit on small ones.

Requires regularization techniques (dropout, batch normalization)

**Conclusion**

In this mini project, implemented and compared Machine Learning (ML) and Deep Learning (DL) models for both regression (diabetes prediction) and classification (heart disease prediction) tasks.

For regression, the ML model (Linear Regression) provided a fast and interpretable solution, but the DL model (Neural Network) performed better in terms of Mean Squared Error (MSE) and R² score. This shows that DL is more effective in capturing complex and non-linear patterns in the data.

For classification, the ML model (Random Forest) achieved high accuracy and interpretability, but the DL model (Neural Network) slightly outperformed it in accuracy and F1-score, demonstrating DL’s advantage in detecting hidden relationships between features.

**Overall comparison:**

ML models are simple, fast, and suitable for smaller datasets where interpretability is important.

DL models require more data, computation, and tuning but generally provide better performance, especially on complex problems.

**Step 1: Define Problem Statement**

Choose one regression problem and one classification problem. Example options:

**- Regression Problem:** Predicting House Prices based on features (size, location, rooms, etc.)

**- Classification Problem:** Email Spam Detection (classify email as spam or ham)

**-** **Alternative Options:**

**- Regression →** Diabetes prediction (continuous values like glucose level)

**- Classification →** Heart disease prediction (Yes/No risk)

**Final Statement Example:**

"The project aims to build both ML and DL models for regression (house price prediction) and classification (spam email detection), and compare their performance in terms of accuracy, error metrics, and computational efficiency."

**Step 2: Create Project Plan & Product Backlog**

|  |  |  |
| --- | --- | --- |
| Week | Task | Deliverable |
| 1 | Define problem statement & collect dataset | Dataset finalized |
| 2 | Preprocess data (cleaning, splitting) | Processed data ready |
| 3 | Build ML models (Regression & Classification) | ML baseline models |
| 4 | Build DL models (Regression & Classification with TensorFlow/Keras) | DL models |
| 5 | Evaluate & Compare ML vs DL | Performance report |
| 6 | Documentation & Final Presentation | Project Report + GitHub repo |

**Product Backlog (Agile Format)**

**User Stories:**

1. As a data scientist, I want to collect and clean datasets so that models can be trained effectively.

2. As a developer, I want to build ML models so I can compare them with DL models.

3. As a researcher, I want to analyze performance so I can evaluate trade-offs between ML and DL.

4. As a student, I want to document the project so I can present clear results.

**Backlog Tasks:**

- Dataset collection & preprocessing

- ML model implementation (Linear Regression, Logistic Regression, etc.)

- DL model implementation (ANN with TensorFlow/Keras)

- Model training, testing, and evaluation

- Comparison of results (accuracy, error metrics)

- Write documentation & prepare presentation

**Step 3: Create Git Repository**

1. Install Git on your system if not already installed.

2. Create a new repository on GitHubLog in to GitHub.

Click the “+” button → New repository.

Enter a repository name, choose public/private, and create it.

3. Open your project folder on your computer.

4. Initialize Git in that folder (this makes it a Git repository).

5. Add your project files to the Git staging area.

6. Commit the changes with a message (for example: “First upload”).

7. Connect your local project to the GitHub repository by adding its URL.

8. Push (upload) your project from your computer to GitHub.

9. Refresh your GitHub repository page — your files shouldnow appear

there.

**Here’s how you can set it up:**

**Steps**

Open GitHub and create a new repository → Name it:

ML-vs-DL-Performance-Analysis

Initialize with a **README.md** file.

Clone repo to local system:

git clone https://github.com/sinchugowda04/Activity-10/blob/main/Activity%2010.docx

Inside the repo, create folder structure:

├── data/ # datasets

├── notebooks/ # Jupyter notebooks

├── src/ # source code

├── results/ # evaluation metrics & plots

├── report/ # final report

├── README.md # project description

└── requirements.txt # dependencies

Commit and push changes:

git add .

git commit -m "Initial project setup"

git push origin main