**ACIVITY-10**

**DEEP LEARNING MODEL IMPLEMENTATION AND PERFORMANCE ANALYSIS**

**Regression model using deep learning algorithm**

**Python Code**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

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

from tensorflow import keras

# 1. Load and prepare data

iris\_data = load\_iris()

data = pd.DataFrame(iris\_data.data, columns=iris\_data.feature\_names)

data['target'] = iris\_data.data[:, 2] # Using petal width as the target

# 2. Display the first few rows of the dataset

print(data.head())

# 3. Visualize correlation with a heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm', center=0)

plt.title('Feature Correlation Heatmap')

plt.show()

# 4. Prepare features and target variable

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

y = data['target']

# Split the data into training, validation, and test sets

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

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

# 5. Normalize the data

scaler = StandardScaler()

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

X\_val = scaler.transform(X\_val)

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

# 6. Build the deep learning model

model = keras.Sequential([

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

keras.layers.Dense(32, activation='relu'),

keras.layers.Dense(1) # Output layer for regression

])

# 7. Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# 8. Train the model

history = model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_val, y\_val))

# 9. Evaluate the model

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

mse = mean\_squared\_error(y\_test, y\_pred)

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

# Custom accuracy-like metric: Percentage of predictions within a threshold

threshold = 0.1 # Set your acceptable range

accuracy\_like = np.mean(np.abs(y\_pred.flatten() - y\_test) <= threshold)

# Display results

print(f'Mean Squared Error: {mse:.2f}')

print(f'R-squared Score: {r2:.2f}')

# Optional: Display a few predictions vs actual values

results = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred.flatten()})

print(results.head())

**OUTPUT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Sepal length(cm) | Sepal width(cm) | Petal length(cm) | Petal width(cm) | target |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 1.4 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | 1.4 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 1.3 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | 1.5 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | 1.4 |

[5 rows x 5 columns]

Epoch 1/100

4/4 ━━━━ 1s 65ms/step - loss: 18.0929 - val\_loss: 19.1862

Epoch 2/100

4/4 ━━━━ 0s 12ms/step - loss: 16.7744 - val\_loss: 18.3727

Epoch 3/100

4/4 ━━━━ 0s 12ms/step - loss: 16.3398 - val\_loss: 17.6929

Epoch 4/100

4/4 ━━━━0s 13ms/step - loss: 15.1802 - val\_loss: 17.0941

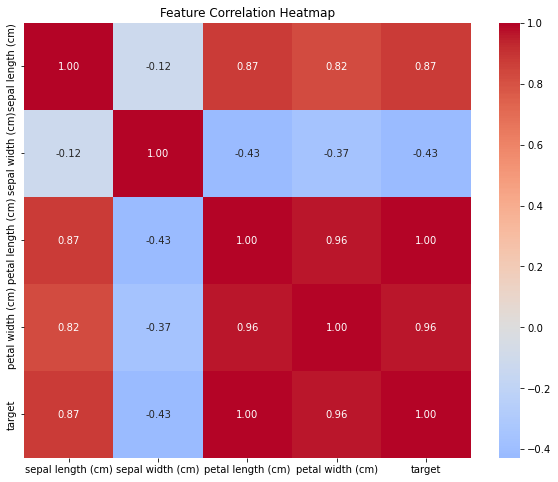
Epoch 5/100

4/4 ━━━━0s 12ms/step - loss: 15.0142 - val\_loss: 16.5433

Mean Squared Error: 0.04

R-squared Score: 0.99

|  |  |  |
| --- | --- | --- |
|  | Actual | Predicted |
| 143 | 5.9 | 5.920325 |
| 56 | 4.7 | 4.306509 |
| 128 | 5.6 | 5.640619 |
| 69 | 2.9 | 3.671755 |
| 68 | 4.4 | 4.786403 |

****

**Regression model using Machine Learning algorithm**

**Python Code**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

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

# 1. Load and prepare data

iris\_data = load\_iris()

data = pd.DataFrame(iris\_data.data, columns=iris\_data.feature\_names)

data['target'] = iris\_data.data[:, 2] # Using petal width as the target

# 2. Display the first few rows of the dataset

print(data.head())

# 3. Visualize correlation with a heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm', center=0)

plt.title('Feature Correlation Heatmap')

plt.show()

# 4. Prepare features and target variable

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

y = data['target']

# Split the data into training and test sets

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

# 5. Normalize the data

scaler = StandardScaler()

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

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

# 6. Build the regression model

model = LinearRegression()

# 7. Train the model

model.fit(X\_train, y\_train)

# 8. Make predictions

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

# 9. Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

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

# Custom accuracy-like metric: Percentage of predictions within a threshold

threshold = 0.1 # Set your acceptable range

accuracy\_like = np.mean(np.abs(y\_pred - y\_test) <= threshold) \* 100

print(f'Mean Squared Error: {mse:.2f}')

print(f'R-squared Score: {r2:.2f}')

print(f'Custom Accuracy-like Metric (within ±{threshold}): {accuracy\_like:.2f}%')

# Optional: Display a few predictions vs actual values

results = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print(results.head())

**OUTPUT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Sepal length(cm) | Sepal width(cm) | Petal length(cm) | Petal width(cm) | target |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 1.4 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | 1.4 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 1.3 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | 1.5 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | 1.4 |

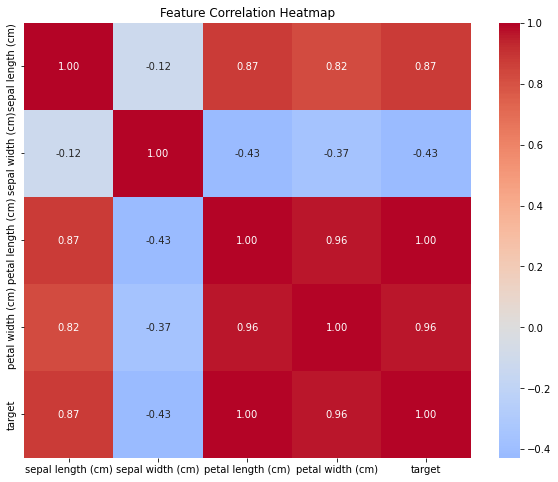
[5 rows x 5 columns]

Mean Squared Error: 0.00

R-squared Score: 1.00

Custom Accuracy-like Metric (within ±0.1): 100.00%

|  |  |  |
| --- | --- | --- |
|  | Actual | Predicted |
| 73 | 4.7 | 4.7 |
| 18 | 1.7 | 1.7 |
| 118 | 6.9 | 6.9 |
| 75 | 4.5 | 4.5 |
| 76 | 4.8 | 4.8 |



**Classification model using Deep Learning algorithm**

**Python Code**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_wine

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

from sklearn.preprocessing import StandardScaler

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

# Load the Wine dataset

data = load\_wine()

X = data.data # Features

y = data.target # Target variable (wine classes)

# One-hot encoding the target variable for multi-class classification

y = to\_categorical(y)

# Split the data into training and testing sets (80% training, 20% testing)

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

# Standardize the feature data (Deep learning models work better with normalized data)

scaler = StandardScaler()

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

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

# Build the deep learning model

model = Sequential()

# Input layer with 13 features, and two hidden layers

model.add(Dense(128, input\_shape=(X\_train.shape[1],), activation='relu')) # First hidden layer

model.add(Dense(64, activation='relu')) # Second hidden layer

# Output layer with 3 neurons for the 3 wine classes and softmax activation for multi-class classification

model.add(Dense(3, activation='softmax'))

# Compile the model

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

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model on test data

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(f'\nTest Accuracy: {test\_acc:.2f}')

# Predictions

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

y\_pred\_classes = np.argmax(y\_pred, axis=1)

y\_test\_classes = np.argmax(y\_test, axis=1)

# Calculate accuracy

accuracy = accuracy\_score(y\_test\_classes, y\_pred\_classes)

print(f'Accuracy: {accuracy:.2f}')

# Confusion matrix and classification report

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test\_classes, y\_pred\_classes))

print("\nClassification Report:")

print(classification\_report(y\_test\_classes, y\_pred\_classes, target\_names=data.target\_names))

# Plotting training and validation accuracy over epochs

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

**OUTPUT:**

runfile('E:/untitled8.py', wdir='E:')

Epoch 1/50

5/5 ━━━2s 119ms/step - accuracy: 0.4734 - loss: 1.0220 - val\_accuracy: 0.8889 - val\_loss: 0.7236

Epoch 2/50

5/5 ━━━ 0s 20ms/step - accuracy: 0.7983 - loss: 0.7303 - val\_accuracy: 0.9722 - val\_loss: 0.5221

Epoch 3/50

5/5 ━━━0s 21ms/step - accuracy: 0.9471 - loss: 0.5433 - val\_accuracy: 1.0000 - val\_loss: 0.3761

Epoch 4/50

5/5 ━━━ 0s 16ms/step - accuracy: 0.9832 - loss: 0.3769 - val\_accuracy: 1.0000 - val\_loss: 0.2728

Epoch 5/50

Test Accuracy: 1.00

2/2 ━━━━ 0s 36ms/step

Accuracy: 1.00

Confusion Matrix:

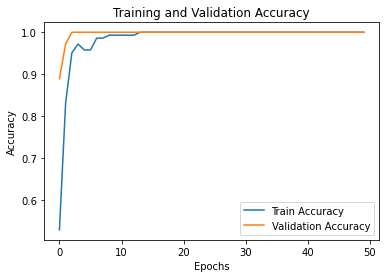
[[14 0 0]

[ 0 14 0]

[ 0 0 8]]

Classification Report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| Class\_0 | 1.00 | 1.00 | 1.00 | 14 |
| Class\_1 | 1.00 | 1.00 | 1.00 | 14 |
| Class\_2 | 1.00 | 1.00 | 1.00 | 8 |
|  |  |  | 1.00 |  |
| accuracy |  |  | 1.00 | 36 |
| Macro avg | 1.00 | 1.00 | 1.00 | 36 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 36 |



**Classification model using Machine Learning algorithm**

Python Code

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_wine

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

from sklearn.linear\_model import LogisticRegression

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

import warnings

warnings.filterwarnings('ignore')

# Load the Wine dataset

data = load\_wine()

X = data.data # Features (chemical composition of wines)

y = data.target # Target variable (wine classes)

# Split the data into training and testing sets (80% training, 20% testing)

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

# Create and fit the Logistic Regression model

model = LogisticRegression(max\_iter=200)

model.fit(X\_train, y\_train)

# Predict on the test set

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

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# Print confusion matrix and classification report

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=data.target\_names))

# Plotting (Optional)

# Visualize the first two features only

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolor='k', s=20)

plt.xlabel(data.feature\_names[0])

plt.ylabel(data.feature\_names[1])

plt.title('Wine Dataset (First Two Features)')

plt.colorbar(label='Wine Class (0, 1, 2)')

plt.show()

**OUTPUT:**

runfile('E:/untitled4.py', wdir='E:')

Accuracy: 0.97

Confusion Matrix:

[[13 1 0]

[ 0 14 0]

[ 0 0 8]]

Classification Report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| Class\_0 | 1.00 | 0.93 | 0.96 | 14 |
| Class\_1 | 0.93 | 1.00 | 0.97 | 14 |
| Class\_2 | 1.00 | 1.00 | 1.00 | 8 |
|  |  |  |  |  |
| accuracy |  |  | 0.97 | 36 |
| Macro avg | 0.98 | 0.98 | 0.98 | 36 |
| Weighted avg | 0.97 | 0.97 | 0.97 | 36 |

