EX. NO: 02

DATE :

MULTILAYER PERCEPTRON WITH HYPERPARAMETER TUNING

AIM:

To build a Multilayer Perceptron (MLP) model using the student-mat.csv dataset and improve its performance through hyperparameter tuning to classify students as pass or fail.

ALGORITHM:

STEP 1: Import required libraries like pandas, NumPy, scikit-learn, TensorFlow, etc.

STEP 2: Load the dataset student-mat.csv and read it into a pandas DataFrame using the appropriate separator (;).

STEP 3: Create a binary classification label: pass (1 if G3 ≥ 10, else 0).

STEP 4: Encode all categorical columns using LabelEncoder.

STEP 5: Drop the original target column G3 (to avoid leakage).

STEP 6: Define the input features X and target label y as pass.

STEP 7: Normalize the features using StandardScaler.

STEP 8: Split the dataset into training and testing sets (80-20 split).

STEP 9: Build the MLP model using Sequential, with multiple dense layers and Dropout to avoid overfitting.

STEP 10: Compile the model with the Adam optimizer and binary_crossentropy loss.

STEP 11: Apply EarlyStopping to prevent overfitting during training.

STEP 12: Train the model using fit() with validation split and early stopping.

STEP 13: Evaluate the model using evaluate() and generate predictions.

STEP 14: Print the classification report and draw the confusion matrix.

PROGRAM:

```
# STEP 1: Install packages
!pip install -q tensorflow pandas scikit-learn

# STEP 2: Import
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# STEP 3: Load dataset
df = pd.read_csv("/content/drive/MyDrive/student-mat.csv", sep=";")

# STEP 4: Binary target column - pass if G3 >= 10
df['pass'] = (df['G3'] >= 10).astype(int)

# STEP 5: Encode categorical columns
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = LabelEncoder().fit_transform(df[col])

# STEP 6: Drop 'G3'
```

```
X = df.drop(['G3', 'pass'], axis=1)
y = df['pass']

# STEP 7: Normalize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# STEP 8: Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# STEP 9: Build model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# STEP 10: Compile
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# STEP 11: Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# STEP 12: Train (FAST)
history = model.fit(
    X_train, y_train,
```

```
    validation_split=0.2,
    epochs=30,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)

# STEP 13: Evaluate
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"\n Final Test Accuracy: {acc * 100:.2f}%")

# STEP 14: Classification report
y_pred = (model.predict(X_test) > 0.5).astype(int)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# STEP 15: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

OUTPUT:

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.4463 - loss: 0.7446 - val_accuracy: 0.5625 - val_loss: 0.7137
Epoch 2/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7027 - loss: 0.6038 - val_accuracy: 0.6406 - val_loss: 0.6741
Epoch 3/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6924 - loss: 0.5932 - val_accuracy: 0.6406 - val_loss: 0.6300
Epoch 4/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7718 - loss: 0.4859 - val_accuracy: 0.6875 - val_loss: 0.5907
Epoch 5/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8024 - loss: 0.4184 - val_accuracy: 0.7500 - val_loss: 0.5526
Epoch 6/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8148 - loss: 0.4154 - val_accuracy: 0.7812 - val_loss: 0.5161
Epoch 7/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8642 - loss: 0.3599 - val_accuracy: 0.7812 - val_loss: 0.4907
Epoch 8/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9315 - loss: 0.2973 - val_accuracy: 0.7969 - val_loss: 0.4802
Epoch 9/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8937 - loss: 0.2966 - val_accuracy: 0.7969 - val_loss: 0.4668
Epoch 10/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8656 - loss: 0.3014 - val_accuracy: 0.7969 - val_loss: 0.4450
Epoch 11/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8851 - loss: 0.2573 - val_accuracy: 0.8125 - val_loss: 0.4328
Epoch 12/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9358 - loss: 0.2031 - val_accuracy: 0.8125 - val_loss: 0.4190
Epoch 13/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9657 - loss: 0.1963 - val_accuracy: 0.8281 - val_loss: 0.4241
Epoch 14/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9729 - loss: 0.1471 - val_accuracy: 0.8125 - val_loss: 0.4167
Epoch 15/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9462 - loss: 0.1632 - val_accuracy: 0.8281 - val_loss: 0.4018
Epoch 16/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9338 - loss: 0.1729 - val_accuracy: 0.8125 - val_loss: 0.3988
Epoch 17/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9684 - loss: 0.1217 - val_accuracy: 0.8281 - val_loss: 0.3948
Epoch 18/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9254 - loss: 0.1610 - val_accuracy: 0.8438 - val_loss: 0.4030
Epoch 19/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9928 - loss: 0.0820 - val_accuracy: 0.8281 - val_loss: 0.4011
Epoch 20/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9462 - loss: 0.1396 - val_accuracy: 0.7969 - val_loss: 0.4024
Epoch 21/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9827 - loss: 0.0898 - val_accuracy: 0.7969 - val_loss: 0.4101
Epoch 22/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9676 - loss: 0.1071 - val_accuracy: 0.7969 - val_loss: 0.4219
```

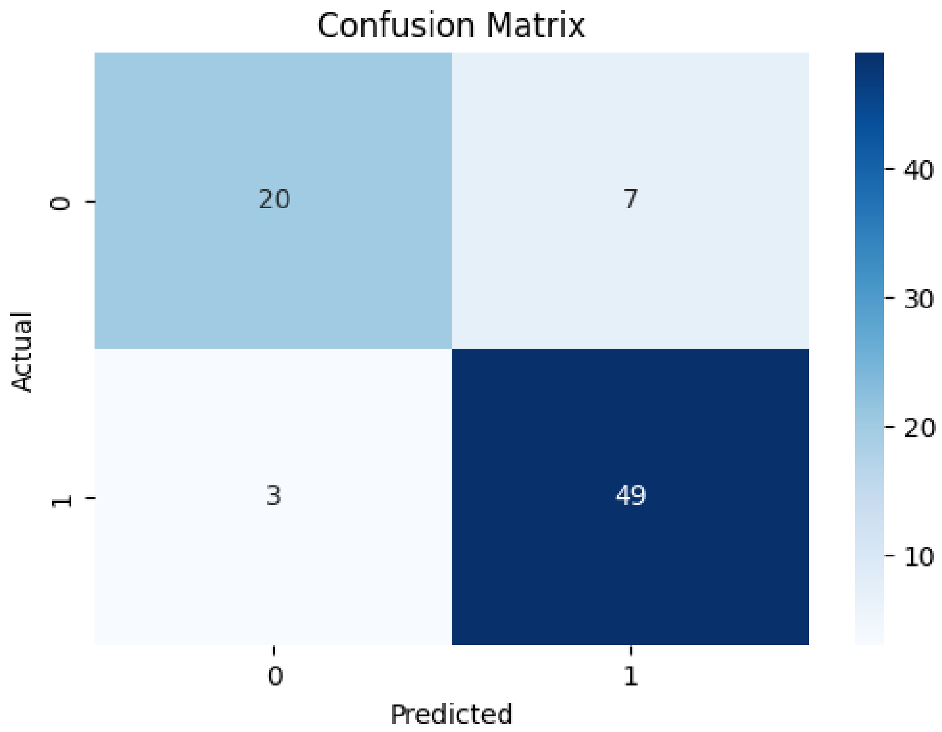
Final Test Accuracy: 87.34%

1/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 50ms/step

3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 30ms/step

Classification Report:

	precision	recall	f1-score	support
0	0.87	0.74	0.80	27
1	0.88	0.94	0.91	52
accuracy			0.87	79
macro avg	0.87	0.84	0.85	79
weighted avg	0.87	0.87	0.87	79



COE (20)	
RECORD (20)	
VIVA (10)	
TOTAL (50)	

RESULT:

The MLP model was successfully trained and tested. It accurately predicted student pass/fail outcomes based on academic and personal features with high classification performance.