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:

```
!pip install -q tensorflow pandas scikit-learn seaborn matplotlib
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 seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

df = pd.read_csv("/content/drive/MyDrive/student-mat.csv", sep=";")

# Create binary target column
df['pass'] = (df['G3'] >= 10).astype(int)

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

# Features and target
X = df.drop(['G3', 'pass'], axis=1)
y = df['pass']

# Standardize features
X_scaled = StandardScaler().fit_transform(X)
```

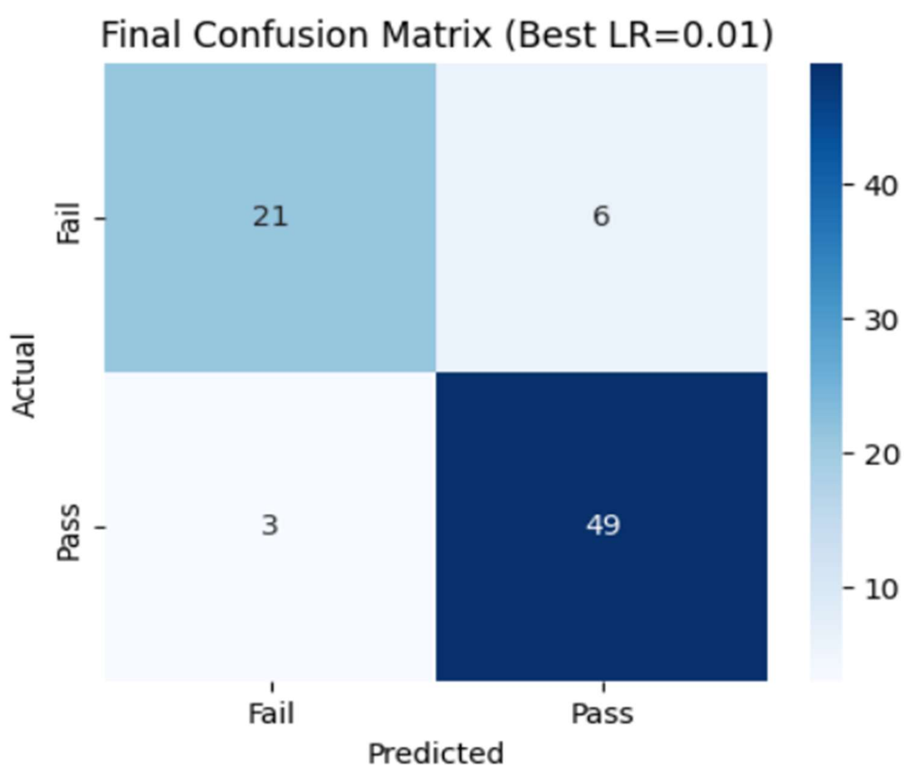
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Custom Callback for test accuracy per epoch
class TestAccuracyCallback(tf.keras.callbacks.Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_accuracies = []

    def on_epoch_end(self, epoch, logs=None):
        test_loss, test_acc = self.model.evaluate(self.test_data[0], self.test_data[1], verbose=0)
        self.test_accuracies.append(test_acc)
learning_rates = [0.0001, 0.001, 0.01, 0.1]
all_summaries = []
best_val_acc = 0
best_model = None
best_lr = 0

for lr in learning_rates:
    print(f"\n Training with Learning Rate: {lr}")

    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')
    ])
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

test_callback = TestAccuracyCallback((X_test, y_test))
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=30,
    batch_size=32,
    callbacks=[early_stop, test_callback],
    verbose=0
)

# Save if best so far
final_val_acc = max(history.history['val_accuracy'])
if final_val_acc > best_val_acc:
    best_val_acc = final_val_acc
    best_model = model
    best_lr = lr
    best_y_pred = (model.predict(X_test) > 0.5).astype("int32")
    best_y_test = y_test

# Save training summary
summary = pd.DataFrame({
    'Epoch': list(range(1, len(history.history['accuracy']) + 1)),
    'Learning Rate': [lr] * len(history.history['accuracy']),
    'Train Accuracy': history.history['accuracy'],
    'Val Accuracy': history.history['val_accuracy'],
```

```
        'Test Accuracy': test_callback.test_accuracies,
        'Train Loss': history.history['loss'],
        'Val Loss': history.history['val_loss'],
    })
    all_summaries.append(summary)
print(f"\n Best Learning Rate: {best_lr}")
print("\nFinal Classification Report:")
print(classification_report(best_y_test, best_y_pred))


cm = confusion_matrix(best_y_test, best_y_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Fail', 'Pass'],
yticklabels=['Fail', 'Pass'])
plt.title(f'Final Confusion Matrix (Best LR={best_lr})')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# STEP 6: Print Combined Summary Table
final_summary = pd.concat(all_summaries, ignore_index=True)
print("\nCombined Training Summary:")
pd.set_option('display.max_rows', None)
print(final_summary.round(4).to_string(index=False))
```

OUTPUT:

```
🔁 Training with Learning Rate: 0.0001
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: Use
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step

🔁 Training with Learning Rate: 0.001
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step

🔁 Training with Learning Rate: 0.01
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step

🔁 Training with Learning Rate: 0.1

🏁 Best Learning Rate: 0.01

📊 Final Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.78      0.82        27
           1       0.89      0.94      0.92        52

    accuracy                           0.89        79
   macro avg       0.88      0.86      0.87        79
weighted avg       0.89      0.89      0.88        79
```



precision recall f1-score support

Combined Training Summary:

Epoch	Learning Rate	Train Accuracy	Val Accuracy	Test Accuracy	Train Loss	Val Loss
1	0.0001	0.5198	0.5312	0.5443	0.6944	0.6894
2	0.0001	0.5714	0.5625	0.6076	0.6769	0.6760
3	0.0001	0.6230	0.5312	0.5949	0.6577	0.6646
4	0.0001	0.6944	0.5625	0.6203	0.6273	0.6546
5	0.0001	0.6230	0.6094	0.6456	0.6408	0.6457
6	0.0001	0.6746	0.6250	0.6456	0.6258	0.6385
7	0.0001	0.6984	0.6875	0.6709	0.6073	0.6316
8	0.0001	0.7103	0.7031	0.6835	0.5922	0.6257
9	0.0001	0.7421	0.6875	0.6962	0.5871	0.6198
10	0.0001	0.7460	0.7031	0.6962	0.5793	0.6143
11	0.0001	0.7460	0.7031	0.6962	0.5724	0.6094
12	0.0001	0.7024	0.7344	0.6709	0.5737	0.6046
13	0.0001	0.7341	0.7344	0.6709	0.5704	0.5999
14	0.0001	0.7500	0.7344	0.6709	0.5672	0.5957
15	0.0001	0.7500	0.7344	0.6709	0.5556	0.5917
16	0.0001	0.7698	0.7188	0.6835	0.5291	0.5872
17	0.0001	0.7579	0.7188	0.6835	0.5339	0.5828
18	0.0001	0.7579	0.7188	0.6835	0.5219	0.5786
19	0.0001	0.7857	0.7188	0.6835	0.5135	0.5748
20	0.0001	0.7738	0.7188	0.6835	0.5154	0.5706
21	0.0001	0.7937	0.7188	0.6835	0.4946	0.5664
22	0.0001	0.7976	0.7188	0.6835	0.5092	0.5622
23	0.0001	0.7857	0.7188	0.6835	0.4973	0.5583
24	0.0001	0.8056	0.7188	0.6835	0.4912	0.5544
25	0.0001	0.8135	0.7188	0.6962	0.4895	0.5501
26	0.0001	0.8016	0.7188	0.6962	0.4817	0.5460
27	0.0001	0.8175	0.7188	0.7089	0.4864	0.5421
28	0.0001	0.7778	0.7188	0.7089	0.4699	0.5375
29	0.0001	0.7857	0.7188	0.7089	0.4697	0.5336
30	0.0001	0.8175	0.7188	0.7089	0.4532	0.5301
1	0.0010	0.5278	0.7031	0.7468	0.6976	0.6006
2	0.0010	0.7183	0.7031	0.7722	0.5474	0.5448
3	0.0010	0.8016	0.7500	0.7975	0.4558	0.5049
4	0.0010	0.7976	0.7500	0.8101	0.4378	0.4744
5	0.0010	0.8532	0.7500	0.8481	0.3638	0.4464
6	0.0010	0.8571	0.7500	0.8481	0.3346	0.4387
7	0.0010	0.8889	0.7812	0.8481	0.3084	0.4299
8	0.0010	0.9008	0.7812	0.8481	0.2783	0.4136
9	0.0010	0.9325	0.7969	0.8608	0.2366	0.4123

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.