**HACKATHON LEVEL-2**

**GEN-AI**

**Githublink :**

**Name : PUGALARASU.S**

**Reg No : 620122114043**

**College Name: AVS ENGINEERING COLLEGE**

**Department : Mechanical engineering**

**Step 1: Import Libraries**

**Start by importing the libraries needed for data processing, modeling, and evaluation.**

**python code**

**# Data processing**

**import pandas as pd**

**import numpy as np**

**# For Machine Learning models**

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

**from sklearn.metrics import accuracy\_score, f1\_score, classification\_report**

**# For Deep Learning**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, LSTM, Embedding, Dropout**

**from tensorflow.keras.preprocessing.text import Tokenizer**

**from tensorflow.keras.preprocessing.sequence import pad\_sequences**

**# Visualization**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**Step 2: Load Data**

**Load your dataset using pandas. Replace 'your\_dataset.csv' with the path to your dataset file.**

**python code**

**# Load dataset**

**df = pd.read\_csv('your\_dataset.csv')**

**# Display first few rows**

**df.head()**

**Step 3: Data Preprocessing**

**Prepare the data for training. For text data, perform tokenization and padding. For numerical data, normalize the values.**

**For Text Data**

**python code**

**# Tokenization for text data**

**tokenizer = Tokenizer(num\_words=5000)**

**tokenizer.fit\_on\_texts(df['text\_column']) # Replace 'text\_column' with your text feature**

**# Convert text to sequences**

**X = tokenizer.texts\_to\_sequences(df['text\_column'])**

**# Padding sequences**

**X = pad\_sequences(X, maxlen=100)**

**# Labels (e.g., binary classification)**

**y = df['label\_column'].values # Replace 'label\_column' with your target feature**

**# Split data 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)**

**For Numerical Data**

**python code**

**from sklearn.preprocessing import StandardScaler**

**# Normalize numerical data**

**scaler = StandardScaler()**

**X = scaler.fit\_transform(df[['numerical\_column']]) # Replace 'numerical\_column' with your feature**

**# Split data 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)**

**Step 4: Choose a Model**

**Define a model. Below is an example of an LSTM-based model for text classification.**

**python code**

**# Define LSTM model**

**model = Sequential()**

**model.add(Embedding(input\_dim=5000, output\_dim=128, input\_length=100))**

**model.add(LSTM(128, return\_sequences=True))**

**model.add(Dropout(0.2))**

**model.add(LSTM(128))**

**model.add(Dense(1, activation='sigmoid')) # Binary classification**

**# Compile the model**

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

**Step 5: Train the Model**

**Train the model using your training data.**

**python code**

**# Train the model**

**history = model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_data=(X\_test, y\_test))**

**Step 6: Evaluate the Model**

**Evaluate the model's performance and calculate metrics like accuracy and F1-score.**

**python code**

**# Predict on the test set**

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

**y\_pred = (y\_pred > 0.5).astype(int) # Convert probabilities to binary output**

**# Calculate metrics**

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

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

**# Display results**

**print(f"Accuracy: {accuracy:.4f}")**

**print(f"F1-score: {f1:.4f}")**

**print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))**

**Visualization of Training History Plot the training and validation loss/accuracy.**

**python code**

**# Plot accuracy and loss over epochs**

**plt.figure(figsize=(12, 4))**

**# Accuracy plot**

**plt.subplot(1, 2, 1)**

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

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

**plt.title('Accuracy over Epochs')**

**plt.xlabel('Epochs')**

**plt.ylabel('Accuracy')**

**plt.legend()**

**# Loss plot**

**plt.subplot(1, 2, 2)**

**plt.plot(history.history['loss'], label='Training Loss')**

**plt.plot(history.history['val\_loss'], label='Validation Loss')**

**plt.title('Loss over Epochs')**

**plt.xlabel('Epochs')**

**plt.ylabel('Loss')**

**plt.legend()**

**plt.show()**

**Step 7: Save the Notebook**

**Save the notebook in Jupyter or Colab after verifying all steps are working correctly.**

**python code**

**# Save the trained model if needed**

**model.save('text\_classification\_model.h5')**

**# Optionally save tokenizer for future use**

**import pickle**

**with open('tokenizer.pkl', 'wb') as f:**

**pickle.dump(tokenizer, f)**

**Output Example**

**After running the code, you'll get:**

**Training history (accuracy/loss over epochs)**

**Evaluation metrics (accuracy, F1-score, classification report)**

**Saved model and tokenizer for reuse.**

**The classification report might look like this:**

**markdown**

**Copy code**

**precision recall f1-score support**

**0 0.95 0.93 0.94 500**

**1 0.92 0.95 0.93 500**

**accuracy 0.94 1000**

**macro avg 0.93 0.94 0.93 1000**

**weighted avg 0.93 0.94 0.93 1000**

**This indicates high performance of the custom LSTM model for the task.**