Code 1

|  |
| --- |
| !pip install pennylane |

Code 2

|  |
| --- |
| import pennylane as qml  from pennylane import numpy as np  from pennylane.optimize import AdamOptimizer  from sklearn.model\_selection import train\_test\_split  from sklearn.feature\_extraction.text import TfidfVectorizer  import pandas as pd  from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score  import math |

Code 3

|  |
| --- |
| # Define quantum device  num\_qubits = 4  num\_layers = 2  dev = qml.device("default.qubit", wires=num\_qubits) |

Code 4

|  |
| --- |
| # Quantum circuit functions  def statepreparation(x):  qml.BasisEmbedding(x, wires=range(0, num\_qubits))  def layer(W):  for i in range(num\_qubits):  qml.Rot(W[i, 0], W[i, 1], W[i, 2], wires=i)  for i in range(num\_qubits):  qml.CNOT(wires=[i, (i + 1) % num\_qubits])  @qml.qnode(dev, interface="autograd")  def circuit(weights, x):  statepreparation(x)  for W in weights:  layer(W)  return qml.expval(qml.PauliZ(0))  def variational\_classifier(weights, bias, x):  return circuit(weights, x) + bias  def square\_loss(labels, predictions):  loss = 0  for l, p in zip(labels, predictions):  loss = loss + (l - p) \*\* 2  return loss / len(labels)  def accuracy(labels, predictions):  return np.mean(np.abs(labels - predictions) < 1e-5)  def cost(weights, bias, X, Y):  predictions = [variational\_classifier(weights, bias, x) for x in X]  return square\_loss(Y, predictions) |

Code 5

|  |
| --- |
| # Load the mail dataset  df\_mail = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/mail\_data.csv') |

Code 6

|  |
| --- |
| # Preprocessing: map 'Category' to binary (1 for spam, 0 for ham)  df\_mail['Category'] = df\_mail['Category'].map(lambda x: 1 if x == 'spam' else 0) |

Code 6a

|  |
| --- |
| # Plot the distribution of categories  plt.figure(figsize=(8, 6))  sns.countplot(x='Category', data=df\_mail, palette='viridis')  plt.title('Distribution of Email Categories')  plt.xlabel('Category')  plt.ylabel('Count')  plt.show() |

Code 7

|  |
| --- |
| # Use TF-IDF to convert 'Message' column into numerical features  tfidf = TfidfVectorizer(max\_features=num\_qubits) # Adjust the number of features to match num\_qubits  X = tfidf.fit\_transform(df\_mail['Message']).toarray() |

Code 8

|  |
| --- |
| # Binarize the TF-IDF values (set threshold at 0.5)  X = np.where(X > 0.5, 1, 0) |

Code 9

|  |
| --- |
| # Labels (target)  y = df\_mail['Category'] |

Code 10

|  |
| --- |
| # Split the data into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.10, random\_state=42, stratify=y) |

Code 11

|  |
| --- |
| # Convert to the required format for PennyLane  X\_train = np.array(X\_train, requires\_grad=False)  Y\_train = np.array(y\_train \* 2 - np.ones(len(y\_train)), requires\_grad=False) |

Code 12

|  |
| --- |
| # Set initial parameters  np.random.seed(0)  weights\_init = 0.01 \* np.random.randn(num\_layers, num\_qubits, 3, requires\_grad=True)  bias\_init = np.array(0.0, requires\_grad=True)  opt = AdamOptimizer(0.125)  num\_it = 70  batch\_size = math.floor(len(X\_train) / num\_it)  weights = weights\_init  bias = bias\_init |

Code 13

|  |
| --- |
| # Training loop  for it in range(num\_it):  batch\_index = np.random.randint(0, len(X\_train), (batch\_size,))  X\_batch = X\_train[batch\_index]  Y\_batch = Y\_train[batch\_index]  weights, bias, \_, \_ = opt.step(cost, weights, bias, X\_batch, Y\_batch)  # Compute accuracy  predictions = [np.sign(variational\_classifier(weights, bias, x)) for x in X\_train]  acc = accuracy(Y\_train, predictions)  print("Iter: {:5d} | Cost: {:0.7f} | Accuracy: {:0.7f}".format(it + 1, cost(weights, bias, X\_train, Y\_train), acc)) |

Code 14

|  |
| --- |
| # Testing phase  X\_test = np.array(X\_test, requires\_grad=False)  Y\_test = np.array(y\_test \* 2 - np.ones(len(y\_test)), requires\_grad=False)  predictions = [np.sign(variational\_classifier(weights, bias, x)) for x in X\_test]  print("Test Accuracy:", accuracy\_score(Y\_test, predictions))  print("Precision:", precision\_score(Y\_test, predictions))  print("Recall:", recall\_score(Y\_test, predictions))  print("F1 Score:", f1\_score(Y\_test, predictions, average='macro')) |