```
!pip install torch torchvision torchaudio
!pip install torch-geometric
!pip install scikit-learn pandas openpyxl
```

```
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtim
Collecting torch-geometric
  Downloading torch_geometric-2.6.1-py3-none-any.whl.metadata (63 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.1/63.1 kB 1.6 MB/s eta 0:00:00
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (3.11.15)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (2025.3.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (2.0.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (5.9.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (3.2.3)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from torch-geometric) (4.67.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch-geometric) (2.
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch-geometric) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch-geometric) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch-geometric) (1.6.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch-geometric) (6.4.3)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch-geometric) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch-geometric) (1.20.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch-geometric) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->torch-geometric) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->torch-geometric) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->torch-geometric) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->torch-geometric) (2025.1
Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 15.5 MB/s eta 0:00:00
Installing collected packages: torch-geometric
Successfully installed torch-geometric-2.6.1
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: openpyxl in /usr/local/lib/python3.11/dist-packages (3.1.5)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.11/dist-packages (from openpyxl) (2.0.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import torch
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv
from sklearn.model_selection import train_test_split

# Load Excel file
df = pd.read_excel("/content/Labeled_USElectionTweets.xlsx")
df = df.dropna(subset=['clean_text', 'sentiment']).copy()

# Encode labels (negative: 0, neutral: 1, positive: 2)
label_encoder = LabelEncoder()
```

```
df['label'] = label_encoder.fit_transform(df['sentiment'])

# Vectorize text using TF-IDF
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(df['clean_text']).toarray()
y = df['label'].values
```

```
# Cosine similarity graph construction (k-NN style)
similarity_matrix = cosine_similarity(X)
threshold = 0.6  # similarity threshold
edge_index = []

for i in range(len(similarity_matrix)):
    for j in range(i+1, len(similarity_matrix)):
        if similarity_matrix[i][j] > threshold:
            edge_index.append([i, j])
            edge_index.append([j, i])  # undirected graph

edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()

# Convert to PyTorch Geometric Data object
x = torch.tensor(X, dtype=torch.float)
y = torch.tensor(y, dtype=torch.long)
data = Data(x=x, edge_index=edge_index, y=y)
```

```
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
import torch.nn as nn

class GCN(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(num_features, 64)
        self.conv2 = GCNConv(64, num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

```
# Split indices manually
train_idx, test_idx = train_test_split(np.arange(len(y)), test_size=0.2, stratify=y, random_state=42)
train_mask = torch.zeros(len(y), dtype=torch.bool)
test_mask = torch.zeros(len(y), dtype=torch.bool)
train_mask[train_idx] = True
test_mask[test_idx] = True
data.train_mask = train_mask
data.test_mask = test_mask

# Model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GCN(num_features=x.shape[1], num_classes=3).to(device)
data = data.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

# Training loop
for epoch in range(1, 201):
    model.train()
    optimizer.zero_grad()
    out = model(data)
    loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()

    if epoch % 20 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
```

```
Epoch 20, Loss: 0.3826
Epoch 40, Loss: 0.2123
```

```
Epoch 60, Loss: 0.1436
Epoch 80, Loss: 0.1099
Epoch 100, Loss: 0.0927
Epoch 120, Loss: 0.0817
Epoch 140, Loss: 0.0741
Epoch 160, Loss: 0.0685
Epoch 180, Loss: 0.0642
Epoch 200, Loss: 0.0610
```

```
model.eval()
_, pred = model(data).max(dim=1)
correct = int(pred[data.test_mask].eq(data.y[data.test_mask]).sum())
acc = correct / int(data.test_mask.sum())
print(f'Test Accuracy: {acc:.4f}')
```

```
Test Accuracy: 0.7584
```

```
from sklearn.metrics import classification_report

y_true = data.y[data.test_mask].cpu().numpy()
y_pred = pred[data.test_mask].cpu().numpy()
print(classification_report(y_true, y_pred, target_names=label_encoder.classes_))
```

```
              precision    recall  f1-score   support

    negative       0.83      0.88      0.85       304
     neutral       0.51      0.49      0.50        92
    positive       0.67      0.27      0.39        22

    accuracy                           0.76       418
   macro avg       0.67      0.55      0.58       418
weighted avg       0.75      0.76      0.75       418
```

```
# ----------------------------------------------
# Traditional ML Models (LogReg, SVM, RF)
# ----------------------------------------------
```

```
# 📌 Install required packages
!pip install -q scikit-learn pandas openpyxl

# 📥 Load data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Load dataset
df = pd.read_excel("/content/Labeled_USElectionTweets.xlsx")
df = df.dropna(subset=['clean_text', 'sentiment'])

# Encode target
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['sentiment'])

# TF-IDF vectorization
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['clean_text']).toarray()
y = df['label']

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Logistic Regression
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
```

```
print(" 🔹 Logistic Regression")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr, target_names=label_encoder.classes_))
```

```
🔹 Logistic Regression
Accuracy: 0.7272727272727273
              precision    recall  f1-score   support

    negative       0.73      0.98      0.84       296
     neutral       0.62      0.14      0.23        92
    positive       0.00      0.00      0.00        30

    accuracy                           0.73       418
   macro avg       0.45      0.37      0.36       418
weighted avg       0.66      0.73      0.65       418

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# Support Vector Machine (SVM)
svm_model = SVC()
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)

print(" 🔹 Support Vector Machine (SVM)")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm, target_names=label_encoder.classes_))
```

```
🔹 Support Vector Machine (SVM)
Accuracy: 0.7416267942583732
              precision    recall  f1-score   support

    negative       0.73      1.00      0.85       296
     neutral       0.93      0.15      0.26        92
    positive       0.00      0.00      0.00        30

    accuracy                           0.74       418
   macro avg       0.56      0.38      0.37       418
weighted avg       0.73      0.74      0.66       418

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print(" 🔹 Random Forest")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf, target_names=label_encoder.classes_))
```

```
🔹 Random Forest
Accuracy: 0.7440191387559809
              precision    recall  f1-score   support

    negative       0.74      0.99      0.85       296
     neutral       0.71      0.16      0.27        92
    positive       1.00      0.13      0.24        30

    accuracy                           0.74       418
   macro avg       0.82      0.43      0.45       418
weighted avg       0.76      0.74      0.68       418
```

```
import torch
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GATConv

class GATNet(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, heads=1):
        super(GATNet, self).__init__()
        self.conv1 = GATConv(input_dim, hidden_dim, heads=heads)
        self.conv2 = GATConv(hidden_dim * heads, output_dim, heads=1)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.elu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

# Create model
gat_model = GATNet(input_dim=1000, hidden_dim=64, output_dim=3).to(device)

# Define optimizer and loss
optimizer = torch.optim.Adam(gat_model.parameters(), lr=0.01, weight_decay=5e-4)
criterion = nn.NLLLoss()

# Training Loop (same style as GCN)
gat_model.train()
for epoch in range(1, 201):
    optimizer.zero_grad()
    out = gat_model(data)
    loss = criterion(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()

    if epoch % 20 == 0:
        print(f'Epoch {epoch}, Loss: {loss.item():.4f}')
```

```
Epoch 20, Loss: 0.3350
Epoch 40, Loss: 0.1867
Epoch 60, Loss: 0.1397
Epoch 80, Loss: 0.1168
Epoch 100, Loss: 0.1024
Epoch 120, Loss: 0.0925
Epoch 140, Loss: 0.0850
Epoch 160, Loss: 0.0786
Epoch 180, Loss: 0.0740
Epoch 200, Loss: 0.0705
```

```
gat_model.eval()
pred = gat_model(data).argmax(dim=1)
correct = pred[data.test_mask] == data.y[data.test_mask]
accuracy = int(correct.sum()) / int(data.test_mask.sum())
print(f"GAT Test Accuracy: {accuracy:.4f}")

# Classification report
from sklearn.metrics import classification_report
print(classification_report(data.y[data.test_mask].cpu(), pred[data.test_mask].cpu(), target_names=label_encoder.classes_))
```

```
GAT Test Accuracy: 0.7321
              precision    recall  f1-score   support

    negative       0.82      0.84      0.83       304
     neutral       0.46      0.49      0.47        92
    positive       0.60      0.27      0.38        22

    accuracy                           0.73       418
   macro avg       0.63      0.53      0.56       418
weighted avg       0.73      0.73      0.73       418
```