**SUMMARY MAKER USING PYTORCH**

**import torch**

**import torch.nn as nn**

**import torch.optim as optim**

**from sklearn.feature\_extraction.text import TfidfVectorizer**

**import numpy as np**

**# Define a simple PyTorch model for scoring sentences**

**class SentenceScorer(nn.Module):**

**def \_\_init\_\_(self, input\_size):**

**super(SentenceScorer, self).\_\_init\_\_()**

**self.fc = nn.Linear(input\_size, 1) # Linear layer for scoring**

**self.sigmoid = nn.Sigmoid() # Sigmoid to map scores between 0 and 1**

**def forward(self, x):**

**x = self.fc(x)**

**x = self.sigmoid(x)**

**return x**

**# Function to preprocess and vectorize sentences**

**def preprocess\_and\_vectorize(text, vectorizer=None):**

**sentences = text.split(". ")**

**if sentences[-1].endswith("."):**

**sentences[-1] = sentences[-1][:-1]**

**if vectorizer is None: # During training**

**vectorizer = TfidfVectorizer(stop\_words="english")**

**tfidf\_matrix = vectorizer.fit\_transform(sentences).toarray()**

**return sentences, tfidf\_matrix, vectorizer**

**else: # During summarization**

**tfidf\_matrix = vectorizer.transform(sentences).toarray()**

**return sentences, tfidf\_matrix**

**# Summarization function**

**def summarize(text, model, vectorizer, top\_k=2):**

**sentences, tfidf\_matrix = preprocess\_and\_vectorize(text, vectorizer)**

**data = torch.tensor(tfidf\_matrix, dtype=torch.float32)**

**with torch.no\_grad():**

**scores = model(data).squeeze(1).numpy()**

**ranked\_sentences = [sentence for \_, sentence in sorted(zip(scores, sentences), reverse=True)]**

**summary = ". ".join(ranked\_sentences[:top\_k])**

**return summary**

**# Training function**

**def train\_model(data, labels, input\_size, epochs=100, learning\_rate=0.01):**

**model = SentenceScorer(input\_size)**

**criterion = nn.BCELoss() # Binary cross-entropy loss for relevance scoring**

**optimizer = optim.Adam(model.parameters(), lr=learning\_rate)**

**for epoch in range(epochs):**

**optimizer.zero\_grad()**

**outputs = model(data)**

**loss = criterion(outputs, labels)**

**loss.backward()**

**optimizer.step()**

**if (epoch + 1) % 10 == 0:**

**print(f"Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}")**

**return model**

**# Main program**

**if \_\_name\_\_ == "\_\_main\_\_":**

**# Step 1: User inputs their notes**

**user\_notes = input("Enter your notes (separate sentences with periods):\n")**

**# Step 2: Split sentences and preprocess notes**

**sentences, tfidf\_matrix, vectorizer = preprocess\_and\_vectorize(user\_notes)**

**# Step 3: Create training data (assign labels manually for simplicity)**

**# IMPORTANT: Replace this with a proper dataset for real applications.**

**# For demonstration purposes, we assume that the first and last sentences are important (label=1).**

**labels = torch.tensor([1 if i == 0 or i == len(sentences) - 1 else 0 for i in range(len(sentences))],**

**dtype=torch.float32).unsqueeze(1)**

**data = torch.tensor(tfidf\_matrix, dtype=torch.float32)**

**# Step 4: Train the model**

**input\_size = tfidf\_matrix.shape[1]**

**model = train\_model(data, labels, input\_size)**

**# Step 5: Generate a summary**

**top\_k = int(input("\nHow many sentences would you like in the summary? "))**

**summary = summarize(user\_notes, model, vectorizer, top\_k=top\_k)**

**print("\nSummary:")**

**print(summary)**