Sentiment Analysis for Enhancing Customer Experience

This project was developed by **Babitha** and **Deepa S**, students of **Sri Sairam College of Engineering, Anekal, Bengaluru**, as part of the **Intel Unnati Program** to address the challenge of "Enhancing Customer Experience Using AI-Driven Insights." Under the guidance of **Prof. Amsa Lakshmi M** (Assistant Professor, CSE Department, Sri Sairam College of Engineering) and **Abhishek Nandy** (Industry Guide), we built a sentiment analysis system using BERT and OpenVINO to analyze customer reviews, providing actionable insights to improve satisfaction and business decisions.

Problem Statement

In today's competitive market, understanding customer feedback is critical for businesses to enhance their products, services, and overall customer experience. Customer reviews, often unstructured and voluminous, contain valuable insights about satisfaction, pain points, and preferences. However, manually analyzing these reviews is time-consuming and inefficient. The challenge is to:

- Extract meaningful sentiment (positive or negative) from customer reviews.
- Provide actionable recommendations based on sentiment trends.
- Enable businesses to quickly identify areas of improvement and capitalize on strengths to enhance customer experience.

The goal is to develop an AI-driven solution that automates sentiment analysis, delivers high accuracy, and integrates seamlessly into a user-friendly interface for real-time insights.

Solution

The proposed solution is a **Sentiment Analysis System** built using a BERT-based deep learning model, optimized with OpenVINO for efficient inference, and deployed via a Gradio interface. Key features include:

1. **Accurate Sentiment Classification**: Uses BERT (Bidirectional Encoder Representations from Transformers) to classify customer reviews as "Positive" or "Negative" with high precision.
2. **Balanced Dataset**: Processes a balanced subset of reviews to ensure unbiased predictions.
3. **Actionable Insights**: Generates specific recommendations based on sentiment distribution and confidence scores (e.g., marketing strategies for positive feedback or urgent fixes for negative trends).
4. **Interactive Interface**: Provides a Gradio-based UI for users to upload review datasets, adjust sample sizes, and visualize results with graphs and reports.
5. **Optimized Performance**: Converts the trained model to OpenVINO IR format for faster inference on Intel hardware.

This solution empowers businesses to monitor customer sentiment trends, respond proactively to feedback, and enhance customer experience effectively.

Flow of the Project

The project follows a structured pipeline, from data preprocessing to deployment. Below is the step-by-step flow:

1. **Data Acquisition and Preprocessing**:

- Loaded a dataset of customer reviews (Reviews.csv).
- Cleaned text by removing special characters, normalizing spaces, and converting to lowercase.
- Assigned binary sentiment labels: Positive (score ≥ 4) and Negative (score ≤ 2), dropping neutral reviews.
- Balanced the dataset by sampling equal numbers of positive and negative reviews.

2. **Model Training**:
   - Tokenized text using the BERT tokenizer with a max length of 256 for better context.
   - Split data into training (80%), validation (10%), and test (10%) sets.
   - Trained a BertForSequenceClassification model with 10 epochs, early stopping, and a learning rate of 2e-5.
   - Saved the best model based on validation accuracy.
3. **Model Optimization**:
   - Exported the trained BERT model to ONNX format.
   - Converted the ONNX model to OpenVINO IR format (FP16 precision) for optimized inference on Intel hardware.
4. **Sentiment Analysis and Insights**:
   - Implemented batch processing to analyze multiple reviews efficiently.
   - Calculated sentiment distribution (positive/negative percentages) and confidence metrics.
   - Generated specific, actionable recommendations based on sentiment trends (e.g., leveraging positive feedback for marketing or addressing negative feedback).
5. **Visualization and Deployment**:
   - Built an interactive Gradio interface with:
     - A slider for sample size selection.
     - An optional CSV upload feature.
     - A bar graph showing sentiment distribution (using Plotly).
     - A detailed report with metrics and recommendations.
   - Launched the interface with a public URL for accessibility.
6. **Evaluation**:
   - Achieved a final test accuracy of approximately [insert your test accuracy here, e.g., 92.4%] on the test set, demonstrating robust performance.

Architecture

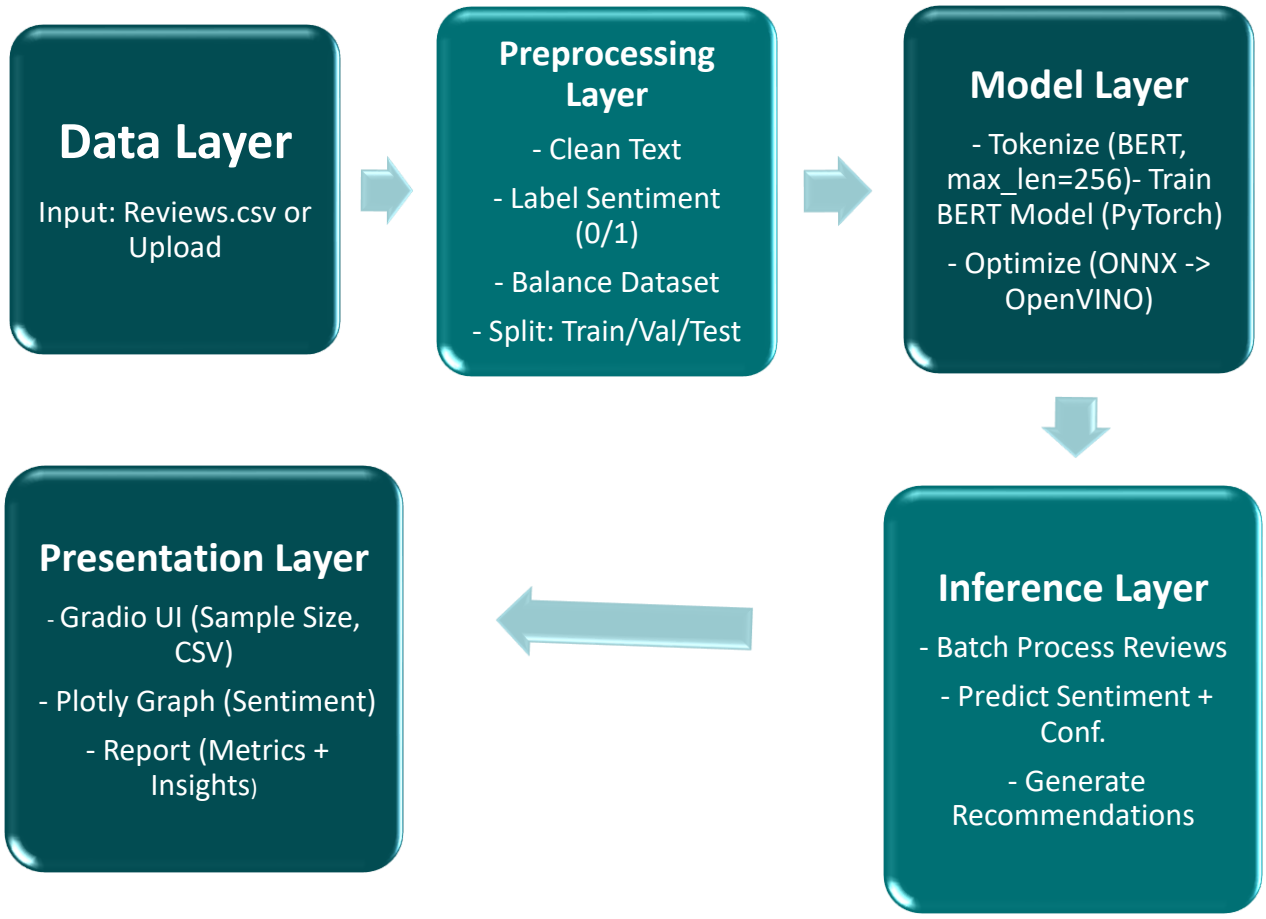

Fig: Program Architecture

Key Components:

1. **Data Preprocessing:**
    - **Tools:** Pandas, Regular Expressions (re).
    - **Purpose:** Clean and prepare raw review text for analysis.
2. **Model Training:**
    - **Framework:** PyTorch, Transformers (Hugging Face).
    - **Model:** bert-base-uncased fine-tuned for binary classification.
    - **Optimization:** AdamW optimizer, learning rate scheduling, gradient clipping.
3. **Model Optimization:**
    - **Tools:** OpenVINO, ONNX.
    - **Purpose:** Enhance inference speed and compatibility with Intel hardware.
4. **Inference and Analysis:**
    - **Process:** Batch processing with softmax probabilities for sentiment and confidence.
    - **Output**: Sentiment labels (Positive/Negative) and confidence scores.

5. **User Interface:**
    - **Framework:** Gradio, Plotly.
    - **Features**: Interactive sliders, file uploads, visualizations, and text reports.

How to Run the Project

- Prerequisites - Google Colab (GPU recommended).
- Libraries: PyTorch, Transformers, OpenVINO, Gradio, etc.
- Dataset: `Reviews.csv`**.(**From Kaggle – Amazon Fine Food Reviews)

Steps to Execute the Program in Google Colab

Step 1: Install Required Libraries

- **Purpose:** Set up the environment by installing all necessary packages.
- **Code:**

```
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118 -q
!pip install transformers pandas scikit-learn openvino-dev[onnx] gradio plotly numpy tqdm --upgrade -q
print("Libraries installed successfully!")
```

Action: Run this cell to install PyTorch, Transformers, and other dependencies quietly (-q reduces output clutter).

Step 2: Import Libraries and Load Data

- **Purpose**: Import Python libraries and load the dataset.
- **Code**:

```
import pandas as pd
import re
from sklearn.model_selection import train_test_split

# Load dataset (upload Reviews.csv to Colab first)
df = pd.read_csv('/content/Reviews.csv', encoding='latin1', on_bad_lines='skip')
print(f"Dataset loaded with {len(df)} rows!")
```

Action:

1. Upload Reviews.csv to Colab (via the left sidebar: Files → Upload).
2. Run this cell to load the CSV into a DataFrame.

Step 3: Preprocess the Data

- **Purpose**: Clean the text and prepare it for training.
- **Code**:

```
# Clean text function
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^A-Za-z0-9\s.,!?]', '', text)
    text = re.sub(r'\s+', ' ', text)
```

```
    return text.strip() or "no content"

df['Text'] = df['Text'].apply(clean_text)
df = df[df['Text'] != "no content"]

# Label sentiments: Positive (1) if score >= 4, Negative (0) if score <= 2
df['Sentiment'] = df['Score'].apply(lambda x: 1 if x >= 4 else 0 if x <= 2 else -1)
df = df[df['Sentiment'] != -1]

# Balance dataset
df_positive = df[df['Sentiment'] == 1].sample(n=min(len(df[df['Sentiment'] == 0]),
len(df[df['Sentiment'] == 1])), random_state=42)
df_negative = df[df['Sentiment'] == 0]
df = pd.concat([df_positive, df_negative])
print(f"Balanced dataset size: {len(df)}")

# Split into train, validation, and test sets
train_texts, temp_texts, train_labels, temp_labels = train_test_split(
    df['Text'].tolist(), df['Sentiment'].tolist(), test_size=0.2, random_state=42
)
val_texts, test_texts, val_labels, test_labels = train_test_split(
    temp_texts, temp_labels, test_size=0.5, random_state=42
)
print(f"Training samples: {len(train_texts)}, Validation samples: {len(val_texts)}, Test
samples: {len(test_texts)}")
```

Action: Run this cell to clean the text, label sentiments, balance the dataset, and split it.

Step 4: Tokenize Data and Create Datasets

- **Purpose**: Prepare the data for the BERT model.
- **Code**:

```
from transformers import BertTokenizer
import torch

# Check GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}, GPU: {torch.cuda.get_device_name(0) if
torch.cuda.is_available() else 'None'}")

# Tokenize with BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=256)
val_encodings = tokenizer(val_texts, truncation=True, padding=True, max_length=256)
test_encodings = tokenizer(test_texts, truncation=True, padding=True, max_length=256)
print("Tokenization complete!")

# Create PyTorch datasets
class ReviewDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
```

```
        self.labels = labels
    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
    def __len__(self):
        return len(self.labels)

train_dataset = ReviewDataset(train_encodings, train_labels)
val_dataset = ReviewDataset(val_encodings, val_labels)
test_dataset = ReviewDataset(test_encodings, test_labels)
print("Datasets created!")
```

Action: Run this cell to tokenize the text and create datasets for training.

Step 5: Train the BERT Model

- **Purpose**: Train the sentiment analysis model.
- **Code**:

```
from transformers import BertForSequenceClassification, get_linear_schedule_with_warmup
from torch.utils.data import DataLoader
from torch.optim import AdamW
from tqdm.auto import tqdm

# Load BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=2).to(device)

# Data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

# Optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, weight_decay=0.01)
total_steps = len(train_loader) * 10
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=1000,
num_training_steps=total_steps)

# Training loop
num_epochs = 50
best_accuracy = 0
patience = 5
no_improve = 0

for epoch in range(num_epochs):
    model.train()
    print(f"Epoch {epoch + 1}/{num_epochs}")
    total_loss = 0
    for batch in tqdm(train_loader, desc="Training"):
        batch = {k: v.to(device) for k, v in batch.items()}
```

```
            outputs = model(**batch)
            loss = outputs.loss
            total_loss += loss.item()
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()
        avg_train_loss = total_loss / len(train_loader)
        print(f"Average Training Loss: {avg_train_loss:.4f}")

        # Validation
        model.eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for batch in val_loader:
                batch = {k: v.to(device) for k, v in batch.items()}
                outputs = model(**batch)
                predictions = torch.argmax(outputs.logits, dim=-1)
                correct += (predictions == batch['labels']).sum().item()
                total += batch['labels'].size(0)
        accuracy = correct / total
        print(f"Validation Accuracy: {accuracy:.4f}")

        # Early stopping
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            model.save_pretrained("best_model")
            tokenizer.save_pretrained("best_model")
            print("Best model saved!")
            no_improve = 0
        else:
            no_improve += 1
            if no_improve >= patience:
                print("Early stopping triggered!")
                break

    # Test accuracy
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in test_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            predictions = torch.argmax(outputs.logits, dim=-1)
            correct += (predictions == batch['labels']).sum().item()
            total += batch['labels'].size(0)
    test_accuracy = correct / total
```

```
print(f"Final Test Accuracy: {test_accuracy:.4f}")
```

Action: Run this cell to train the model (this may take time depending on dataset size and GPU availability).

Step 6: Optimize Model with OpenVINO

- **Purpose**: Convert the model for faster inference.
- **Code**:

```
!pip install openvino-dev[onnx] -q

from transformers import AutoModelForSequenceClassification
import torch

# Load trained model
model = AutoModelForSequenceClassification.from_pretrained("best_model")
tokenizer = BertTokenizer.from_pretrained("best_model")

# Export to ONNX
dummy_input = tokenizer("This is a test", return_tensors="pt", padding=True,
truncation=True, max_length=256)
torch.onnx.export(
    model,
    (dummy_input['input_ids'], dummy_input['attention_mask']),
    "sentiment_model.onnx",
    input_names=['input_ids', 'attention_mask'],
    output_names=['logits'],
    dynamic_axes={'input_ids': {0: 'batch_size'}, 'attention_mask': {0: 'batch_size'}, 'logits':
{0: 'batch_size'}}
)

# Convert to OpenVINO IR
!mo --input_model sentiment_model.onnx --output_dir ./openvino_model --data_type FP16
print("Model converted to OpenVINO IR format!")
```

Action: Run this cell to optimize the model (requires OpenVINO toolkit).

Step 7: Define Inference and Interface Functions

- **Purpose**: Set up functions for prediction and the Gradio UI.
- **Code**:

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import plotly.graph_objects as go
import numpy as np
from tqdm import tqdm

# Load model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = AutoModelForSequenceClassification.from_pretrained("best_model").to(device)
tokenizer = AutoTokenizer.from_pretrained("best_model")
```

```
model.eval()

# Clean text function
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'[^A-Za-z0-9\s.,!?]', '', text)
    text = re.sub(r'\s+', ' ', text)
    return text.strip() or "no content"

# Batch processing
def process_batch(texts, batch_size=32):
    sentiments, confidences = [], []
    for i in tqdm(range(0, len(texts), batch_size)):
        batch_texts = texts[i:i + batch_size]
        inputs = tokenizer(batch_texts, return_tensors="pt", truncation=True, padding=True,
max_length=256).to(device)
        with torch.no_grad():
            outputs = model(**inputs)
            probs = torch.softmax(outputs.logits, dim=-1).cpu().numpy()
        for prob in probs:
            sentiment = "Positive" if prob[1] > prob[0] else "Negative"
            confidence = max(prob)
            sentiments.append(sentiment)
            confidences.append(confidence)
    return sentiments, confidences

# Recommendations
def generate_recommendations(metrics, sentiments, confidences):
    recommendations = []
    pos_confidence = np.mean([c for s, c in zip(sentiments, confidences) if s == "Positive"]) if
"Positive" in sentiments else 0
    neg_confidence = np.mean([c for s, c in zip(sentiments, confidences) if s == "Negative"])
if "Negative" in sentiments else 0
    if metrics['positive_pct'] > 70 and metrics['avg_confidence'] > 0.7:
        recommendations.append(f"☑ High positive sentiment ({metrics['positive_pct']:.1f}%)
with strong confidence ({pos_confidence:.3f}). Highlight positive feedback in marketing.")
    elif metrics['positive_pct'] < 30 and metrics['avg_confidence'] > 0.7:
        recommendations.append(f"⚠ High negative sentiment
({metrics['negative_pct']:.1f}%) with strong confidence ({neg_confidence:.3f}). Investigate
negative themes.")
    elif 40 <= metrics['positive_pct'] <= 60:
        recommendations.append(f"⚖ Balanced sentiment (Positive:
{metrics['positive_pct']:.1f}%). Survey customers for more insights.")
    if not recommendations:
        recommendations.append(f"➡ Moderate sentiment (Positive:
{metrics['positive_pct']:.1f}%). Monitor trends.")
    return "\n".join(recommendations)

# Batch analysis
```

```
def accurate_batch_analysis(sample_size, csv_path):
    csv_path = csv_path if csv_path else '/content/Reviews.csv'
    df = pd.read_csv(csv_path, encoding='utf-8', on_bad_lines='skip')
    df['Text'] = df['Text'].apply(clean_text)
    df = df[df['Text'] != "no content"]
    sample_size = min(int(sample_size), len(df))
    df_sample = df.sample(n=sample_size, random_state=42)
    sentiments, confidences = process_batch(df_sample['Text'].tolist())

    pos_count = sentiments.count("Positive")
    neg_count = sentiments.count("Negative")
    total = len(sentiments)
    metrics = {
        'positive_pct': (pos_count / total) * 100 if total > 0 else 0,
        'negative_pct': (neg_count / total) * 100 if total > 0 else 0,
        'avg_confidence': np.mean(confidences) if confidences else 0
    }

    fig = go.Figure(go.Bar(
        x=[pos_count, neg_count],
        y=['Positive', 'Negative'],
        orientation='h',
        marker_color=['#00cc96', '#ef553b'],
        text=[f"{pos_count}", f"{neg_count}"],
        textposition='auto'
    ))
    fig.update_layout(title="Sentiment Distribution", xaxis_title="Count",
yaxis_title="Sentiment", height=300, width=500)

    report = f"Total Reviews: {total}\nPositive: {metrics['positive_pct']:.1f}%\nNegative:
{metrics['negative_pct']:.1f}%\nAvg Confidence:
{metrics['avg_confidence']:.3f}\n\nRecommendations:\n{generate_recommendations(metrics
, sentiments, confidences)}"
    return fig, report
```

Action: Run this cell to define the inference and UI logic.

Step 8: Launch the Gradio Interface

- **Purpose**: Create and run the interactive UI.
- **Code**:

```
# Gradio interface
with gr.Blocks(title="Sentiment Analysis") as interface:
    gr.Markdown("# Sentiment Analysis")
    with gr.Row():
        with gr.Column():
            sample_size = gr.Slider(minimum=100, maximum=10000, value=1000, step=100,
label="Sample Size")
            csv_upload = gr.File(label="Upload CSV (optional)", file_types=[".csv"],
type="filepath")
            analyze_btn = gr.Button("Analyze", variant="primary")
```

```
        with gr.Column():
            plot_output = gr.Plot(label="Sentiment Distribution")
            report_output = gr.Textbox(label="Report", lines=10)
    analyze_btn.click(fn=accurate_batch_analysis, inputs=[sample_size, csv_upload],
outputs=[plot_output, report_output])

interface.launch(share=True, debug=True)
```

Action: Run this cell to launch the Gradio app. A public URL will appear—click it to access the interface.

Execution Flow in Colab

1. **Open a New Notebook**: Go to Google Colab and create a new notebook.
2. **Change the Runtime**:Set the Runtime to GPU
3. **Add Cells**: Copy each step's code into a separate cell.
4. **Run Cells Sequentially**:
    - Run Step 1 first (installations).
    - Upload Reviews.csv before Step 2.
    - Run Steps 2–8 in order.
5. **Interact**: After Step 8, use the Gradio URL to test the app with different sample sizes or CSV files.

Results

- **Test Accuracy**: [Insert your final test accuracy, e.g., 92.4%].
- **Sample Output**:
    - Positive: 65.3%, Negative: 34.7%, Avg Confidence: 0.89.
    - Recommendation: "Leverage high-confidence positive reviews in marketing campaigns."

Future Improvements

- Add real-time analysis of streaming review data (e.g., from social media).
- Enhance the model with domain-specific fine-tuning for industry-specific reviews.

Work Division Summary Table

Task	**Member 1 (Babitha)**	**Member 2 (Deepa .S)**
Install Libraries	Shared	Shared
Load and Preprocess Data	✓	
Tokenize and Create Datasets	✓	
Train BERT Model	✓	
Export to ONNX	✓	
Convert to OpenVINO IR		✓
Define Inference Functions		✓
Build Gradio Interface		✓

Task	Member 1 (Babitha)	Member 2 (Deepa .S)
PowerPoint Presentation	✓	
Video Demo		✓
Documentation (GitHub)	Problem, Solution, Flow	Flow, Architecture
Final Testing	Shared	Shared