Axa Gen AI

SRC

1) **Preprocess_1.py**

```
import os
import re
import json

# Path to dataset directory
DATASET_PATH = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\datasets\Transcripts_v3 -
Dummy Data\transcripts_v3"

# Function to clean customer statements which remove noise
def clean_text(text):
    text = re.sub(r"\[\d{2}:\d{2}:\d{2}\]", "", text)  # Removes timestamps
    text = re.sub(r"[^a-zA-Z0-9.,!? ]+", "", text)  # Keep meaningful characters
    text = text.strip()  # Remove extra spaces
    return text

# Function to extract customer (Member) statements from transcript files
def extract_customer_statements(file_path):
    customer_lines = []
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            for line in file:
                # Match variations of "Member"
                if re.match(r"(?i)^\s*Member\s*:", line):
                    clean_line = re.sub(r"(?i)^\s*Member\s*:", "", line).strip()  # Remove "Member" prefix
                    cleaned_text = clean_text(clean_line)  # Clean extracted text
                    if cleaned_text:  # Only add non-empty statements
                        customer_lines.append(cleaned_text)
    except Exception as e:
        print(f"Error processing file {file_path}: {e}")

    return customer_lines

# Function to process all transcripts and save structured output
def process_transcripts():
    transcripts_data = {}

    for transcript_file in os.listdir(DATASET_PATH):
        file_path = os.path.join(DATASET_PATH, transcript_file)

        # Extract customer statements
```

```
        customer_statements = extract_customer_statements(file_path)

        # Only add files that have valid customer statements
        if customer_statements:
            transcripts_data[transcript_file] = customer_statements
        else:
            print(f"Skipping empty transcript: {transcript_file}")

    # Save processed data to JSON format
    output_path = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\processed_transcripts.json"

    with open(output_path, "w", encoding="utf-8") as json_file:
        json.dump(transcripts_data, json_file, indent=4)

    print(f"Processed data saved to: {output_path}")

# Run the script
if __name__ == "__main__":
    process_transcripts()
```

2) sentiment_analysis_2.py

```
import json
import pandas as pd
from transformers import pipeline

# Load the best available sentiment model
model_name = "cardiffnlp/twitter-roberta-base-sentiment-latest"  # pre-trained sentiment model
sentiment_analyzer = pipeline("text-classification", model=model_name, tokenizer=model_name,
device=0)

# Loading processed transcripts
input_file = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\processed_transcripts.json"
output_file = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\final_classified_results.csv"

with open(input_file, "r", encoding="utf-8") as file:
    transcripts_data = json.load(file)

# Define function for sentiment classification
def classify_sentiment(statement):

    # Classifies sentiment as Positive, Neutral, or Negative based on model confidence scores.

    try:
        result = sentiment_analyzer(statement, truncation=True, max_length=512, batch_size=16)[0]  #
Batch processing
        label = result["label"]
```

```
        # Convert model labels to standard format
        if "positive" in label.lower():
            return "Positive"
        elif "negative" in label.lower():
            return "Negative"
        else:
            return "Neutral"
    except Exception as e:
        print(f"Error processing: {e}")
        return "Neutral"

# Define function to determine call outcome
def determine_call_outcome(statements):

    # Determines if the call issue was resolved based on sentiment distribution.

    positive_count = sum(1 for s in statements if classify_sentiment(s) == "Positive")
    negative_count = sum(1 for s in statements if classify_sentiment(s) == "Negative")

    return "Issue Resolved" if positive_count > negative_count else "Follow-up Needed"

# Process all transcripts
results = []

for transcript_file, customer_statements in transcripts_data.items():
    sentiments = [classify_sentiment(statement) for statement in customer_statements]
    outcome = determine_call_outcome(customer_statements)

    # Store results
    for statement, sentiment in zip(customer_statements, sentiments):
        results.append({
            "Transcript": transcript_file,
            "Customer Statement": statement,
            "Sentiment": sentiment,
            "Call Outcome": outcome
        })

# Convert results to DataFrame and save as CSV
df_results = pd.DataFrame(results)
df_results.to_csv(output_file, index=False)

print(f"Results saved to: {output_file}")
```

3) Resample_3.py

```
import pandas as pd
```

```
from sklearn.utils import resample

# Load the Dataset
file_path = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\final_classified_results.csv"
df = pd.read_csv(file_path)

# Verify Original Label Distribution
print("\n### Original Sentiment Distribution ###")
print(df["Sentiment"].value_counts())

# Separate Sentiment Classes
df_positive = df[df["Sentiment"] == "Positive"]
df_neutral = df[df["Sentiment"] == "Neutral"]
df_negative = df[df["Sentiment"] == "Negative"]

# Upsample Positive and Negative to Match Neutral (859 samples)
df_positive_upsampled = resample(df_positive, replace=True, n_samples=859, random_state=42)
df_negative_upsampled = resample(df_negative, replace=True, n_samples=859, random_state=42)

# Combine & Shuffle the Dataset
df_balanced = pd.concat([df_positive_upsampled, df_neutral, df_negative_upsampled])
df_balanced = df_balanced.sample(frac=1, random_state=42)  # Shuffle dataset

# Verify New Label Distribution
print("\n### Balanced Sentiment Distribution ###")
print(df_balanced["Sentiment"].value_counts())

# Convert Sentiment Labels to Numeric Format
label_mapping = {"Positive": 2, "Neutral": 1, "Negative": 0}
df_balanced["label"] = df_balanced["Sentiment"].map(label_mapping)

# Save the Balanced Dataset for Fine-Tuning
balanced_file_path = r"C:\Uday\Constant\AXA
Insurance\Axa_Gen_AI\results\balanced_final_classified_results.csv"
df_balanced.to_csv(balanced_file_path, index=False)

print(f"\nBalanced dataset saved at: {balanced_file_path}")
```

4) Fine_tune_sentiment_4.py

```
import pandas as pd
import torch
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer,
TrainingArguments, AutoConfig
import os
```

```
# Load and Prepare Data
file_path = r"C:\Uday\Constant\AXA
Insurance\Axa_Gen_AI\results\balanced_final_classified_results.csv"
df = pd.read_csv(file_path)

# Convert Text Labels to Numeric Labels
label_mapping = {"Positive": 2, "Neutral": 1, "Negative": 0}
df["label"] = df["Sentiment"].map(label_mapping)

# Load Tokenizer
model_name = "distilbert-base-uncased" # model
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Tokenize Data
def tokenize_function(examples):
    return tokenizer(examples["Customer Statement"], padding="max_length", truncation=True)

dataset = Dataset.from_pandas(df[["Customer Statement", "label"]])
dataset = dataset.map(tokenize_function, batched=True)

# Split into Train & Validation Sets
train_test_split = dataset.train_test_split(test_size=0.2)
train_dataset = train_test_split["train"]
val_dataset = train_test_split["test"]

# Load Pretrained Model
num_labels = 3  # Sentiment: Positive, Neutral, Negative
config = AutoConfig.from_pretrained(model_name, num_labels=num_labels)
model = AutoModelForSequenceClassification.from_pretrained(model_name, config=config)

# Define Training Arguments
output_dir = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\fine_tuned_sentiment_model"
os.makedirs(output_dir, exist_ok=True)

training_args = TrainingArguments(
    output_dir=output_dir,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=2,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=12,
    learning_rate=5e-6,
    weight_decay=0.01,
    lr_scheduler_type="linear",
    logging_dir="./logs",
    logging_steps=10,
```

```
    fp16=True,
    gradient_accumulation_steps=2,
    gradient_checkpointing=True,
    push_to_hub=False
)

# Create Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

# Train Model
trainer.train()

# Save Fine-Tuned Model
model.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)

print(f"\nFine-tuned model saved at: {output_dir}")
```

5) Generating_groundtruth_label_dataset_5.py

```
import pandas as pd
from sklearn.utils import resample

# Load the Ground Truth Dataset
ground_truth_path = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\ground_truth_dataset.csv"
df_ground_truth = pd.read_csv(ground_truth_path)

# Check Initial Label Distribution
label_counts = df_ground_truth["Sentiment_actual"].value_counts()
print("\nOriginal Label Distribution:\n", label_counts)

# 3: Separate Classes
df_positive = df_ground_truth[df_ground_truth["Sentiment_actual"] == "Positive"]
df_neutral = df_ground_truth[df_ground_truth["Sentiment_actual"] == "Neutral"]
df_negative = df_ground_truth[df_ground_truth["Sentiment_actual"] == "Negative"]

# Adaptive Sampling
# Setting the target size to the class with the lowest count
target_size = len(df_neutral)

df_positive_resampled = resample(df_positive, replace=True, n_samples=target_size, random_state=42)
df_negative_resampled = resample(df_negative, replace=True, n_samples=target_size, random_state=42)
```

```
# Merge Balanced Dataset
df_balanced_ground_truth = pd.concat([df_positive_resampled, df_neutral, df_negative_resampled])
df_balanced_ground_truth = df_balanced_ground_truth.sample(frac=1, random_state=42)  # Shuffle dataset

# Verify the New Label Distribution
balanced_label_counts = df_balanced_ground_truth["Sentiment_actual"].value_counts()
print("\nBalanced Label Distribution:\n", balanced_label_counts)

# Save the New Balanced Ground Truth Dataset
balanced_ground_truth_path = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\balanced_ground_truth_dataset.csv"
df_balanced_ground_truth.to_csv(balanced_ground_truth_path, index=False)

print(f"\nBalanced ground truth dataset saved at: {balanced_ground_truth_path}")
```

6) **Model_evaluation.py**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report
)

# Load Prediction & Ground Truth Data
file_predictions = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\balanced_final_classified_results.csv"
file_ground_truth = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\balanced_ground_truth_dataset.csv"

# Read Data
df_predictions = pd.read_csv(file_predictions)
df_ground_truth = pd.read_csv(file_ground_truth)

# Standardizing Column Names
df_predictions = df_predictions.rename(columns={"Sentiment": "Sentiment_predicted"})
df_ground_truth = df_ground_truth.rename(columns={"Sentiment_actual": "Sentiment"})

# Merge on 'Customer Statement'
df_merged = df_predictions.merge(df_ground_truth, on="Customer Statement", how="inner")

# Extract labels
y_true = df_merged["Sentiment"]
```

```
y_pred = df_merged["Sentiment_predicted"]

# Evaluation Metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average="weighted", zero_division=1)
recall = recall_score(y_true, y_pred, average="weighted", zero_division=1)
f1 = f1_score(y_true, y_pred, average="weighted", zero_division=1)

# Print Key Evaluation Metrics
print("\n**Evaluation Metrics (Fine-Tuned Model)**")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Classification Report
print("\n**Detailed Classification Report**")
print(classification_report(y_true, y_pred, zero_division=1))

# Confusion Matrix
plt.figure(figsize=(7, 6))
conf_matrix = confusion_matrix(y_true, y_pred, labels=["Positive", "Negative", "Neutral"])
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Positive",
"Negative", "Neutral"], yticklabels=["Positive", "Negative", "Neutral"])
plt.xlabel("Predicted Labels")
plt.ylabel("Actual Labels")
plt.title("Confusion Matrix - Fine-Tuned Model")
plt.show()

print("\n**Model Evaluation Completed!**")
```

7) Insights.py

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from collections import Counter

# Load Prediction Data
file_predictions = r"C:\Uday\Constant\AXA
Insurance\Axa_Gen_AI\results\balanced_final_classified_results.csv"
df_predictions = pd.read_csv(file_predictions)

# Debugging Column Names
print("\nDebugging Column Names:")
print(df_predictions.columns)
```

```
# Standardizing Column Names
if "Sentiment" in df_predictions.columns:
    df_predictions = df_predictions.rename(columns={"Sentiment": "Sentiment_predicted"})

# Sentiment Distribution
plt.figure(figsize=(8, 5))
if "Sentiment_predicted" in df_predictions.columns:
    sns.countplot(data=df_predictions, x="Sentiment_predicted", hue="Sentiment_predicted",
palette="coolwarm", legend=False)
    plt.title("Sentiment Distribution in Customer Calls")
    plt.xlabel("Sentiment Category")
    plt.ylabel("Count")
    plt.show()
else:
    print("Column 'Sentiment_predicted' not found.")

# Resolution Rate (Issue Resolved vs. Follow-up Needed)
if "Call Outcome" in df_predictions.columns:
    resolution_counts = df_predictions["Call Outcome"].value_counts(normalize=True) * 100
    plt.figure(figsize=(6, 4))
    sns.barplot(x=resolution_counts.index, y=resolution_counts.values, hue=resolution_counts.index,
palette="coolwarm", legend=False)
    plt.title("Resolution Rate: Issue Resolved vs. Follow-up Needed")
    plt.ylabel("Percentage (%)")
    plt.xlabel("Call Outcome")
    plt.ylim(0, 100)
    plt.show()
else:
    print("Column 'Call Outcome' not found.")

# Sentiment vs. Call Outcome Heatmap ---
if "Call Outcome" in df_predictions.columns and "Sentiment_predicted" in df_predictions.columns:
    plt.figure(figsize=(8, 6))
    sentiment_vs_resolution = pd.crosstab(df_predictions["Sentiment_predicted"], df_predictions["Call
Outcome"], normalize="index") * 100
    sns.heatmap(sentiment_vs_resolution, annot=True, cmap="coolwarm", fmt=".1f")
    plt.title("Sentiment vs. Call Outcome")
    plt.xlabel("Call Outcome")
    plt.ylabel("Predicted Sentiment")
    plt.show()
else:
    print("Missing 'Sentiment_predicted' or 'Call Outcome'.")

# Word Cloud of Negative & Positive Sentiments
if "Customer Statement" in df_predictions.columns and "Sentiment_predicted" in df_predictions.columns:
```

```
    negative_text = " ".join(df_predictions[df_predictions["Sentiment_predicted"] ==
"Negative"]["Customer Statement"])
    positive_text = " ".join(df_predictions[df_predictions["Sentiment_predicted"] ==
"Positive"]["Customer Statement"])

    # Negative Sentiment Word Cloud
    plt.figure(figsize=(8, 5))
    wordcloud_neg = WordCloud(width=800, height=400, background_color="black",
colormap="Reds").generate(negative_text)
    plt.imshow(wordcloud_neg, interpolation="bilinear")
    plt.axis("off")
    plt.title("Word Cloud of Negative Sentiments")
    plt.show()

    # Positive Sentiment Word Cloud
    plt.figure(figsize=(8, 5))
    wordcloud_pos = WordCloud(width=800, height=400, background_color="white",
colormap="Greens").generate(positive_text)
    plt.imshow(wordcloud_pos, interpolation="bilinear")
    plt.axis("off")
    plt.title("Word Cloud of Positive Sentiments")
    plt.show()
else:
    print("Missing 'Customer Statement' or 'Sentiment_predicted'.")

# Top 10 Recurring Customer Complaints
if "Customer Statement" in df_predictions.columns and "Sentiment_predicted" in df_predictions.columns:
    negative_statements = df_predictions[df_predictions["Sentiment_predicted"] ==
"Negative"]["Customer Statement"]
    words = " ".join(negative_statements).split()
    common_words = Counter(words).most_common(10)

    # Plot Top 10 Complaints
    plt.figure(figsize=(8, 5))
    words, counts = zip(*common_words)
    sns.barplot(x=list(counts), y=list(words), hue=list(words), palette="Reds", legend=False)
    plt.xlabel("Frequency")
    plt.ylabel("Words")
    plt.title("Top 10 Recurring Customer Complaints")
    plt.show()
else:
    print("Missing 'Customer Statement'.")

print("\n **Comprehensive insights generation completed!**")
```

Test

1) test_preprocessing.py

```
import json
import os

# Load processed data
processed_file = r"C:\Uday\Constant\AXA Insurance\Axa_Gen_AI\results\processed_transcripts.json"

def test_preprocessing():
    assert os.path.exists(processed_file), "Processed data file not found!"

    with open(processed_file, "r", encoding="utf-8") as file:
        data = json.load(file)

    assert isinstance(data, dict), "Processed data should be a dictionary!"

    for key, statements in data.items():
        assert isinstance(statements, list), f"Customer statements in {key} should be a list!"
        assert all(isinstance(s, str) for s in statements), f"All statements should be strings in {key}!"
        assert all(len(s) > 0 for s in statements), f"Empty statement found in {key}!"

    print("Preprocessing test passed!")

if __name__ == "__main__":
    test_preprocessing()
```

2) test_sentiment.py

```
import pandas as pd

# Load sentiment results
predictions_file = r"C:\Uday\Constant\AXA
Insurance\Axa_Gen_AI\results\balanced_final_classified_results.csv"
df = pd.read_csv(predictions_file)

def test_sentiment():
    valid_labels = {"Positive", "Negative", "Neutral"}

    assert "Sentiment" in df.columns, "Sentiment column missing!"
    assert df["Sentiment"].isin(valid_labels).all(), "Invalid sentiment labels detected!"

    print("Sentiment classification test passed!")

if __name__ == "__main__":
    test_sentiment()
```