## LLM-Powered Text Analysis and Anomaly Detection Pipeline

**Time Limit:** 8 Hours (Do not exceed).

**Submission Deadline:** Submit within 48 hours of receiving this project.

### Task Description

Develop an ML pipeline to analyze text data (e.g., customer feedback, service inquiries, or product descriptions) using LLMs and NLP techniques. Extract insights (entities, sentiment, themes), detect anomalies (e.g., unusual patterns in text), and augment the dataset with synthetic examples using LLMs. Options include Google's Vertex AI with Gemini 2.0 Flash, OpenAI's API with GPT-4o, or LangChain with multiple LLMs (e.g., Claude, LLaMA). Use Python with scikit-learn for anomaly detection, store data locally (CSV or SQLite), and track metadata in JSON. The pipeline is designed for experimentation, minimizing deployment complexity, and is suitable for text-heavy datasets like service requests or listings.

This assignment emphasizes LLM and NLP skills, testing your ability to extract insights, detect anomalies, and augment data using modern frameworks.

### Detailed Steps

#### Step 1: Data Preparation & Augmentation (2 hr)

**Objective:** Prepare a text dataset and augment it with synthetic examples using LLMs.

**Tasks:**

1. **Prepare a Sample Dataset:**
    - Use a synthetic dataset of text entries. Schema: { id: int, text: string, timestamp: string }.

Example Data (CSV, data.csv):

```
id,text,timestamp
1,"Need urgent help with delivery issues","2025-03-27T10:00:00"
2,"Product quality is great, fast shipping","2025-03-27T12:00:00"
3,"Unusually high price for this item!!!","2025-03-27T14:00:00"
```

2. **Augment Data with LLMs:**

**Option 1: Gemini 2.0 Flash (Vertex AI):**

```
import pandas as pd
from vertexai.preview.language_models import TextGenerationModel

df = pd.read_csv("data.csv")
model = TextGenerationModel.from_pretrained("gemini-2.0-flash")
synthetic_texts = []
for text in df["text"]:
    prompt = f"Generate a paraphrase of this text: '{text}'"
    response = model.predict(prompt, max_output_tokens=50).text
    synthetic_texts.append({"id": len(df) + len(synthetic_texts) + 1, "text":
response, "timestamp": "2025-03-27T16:00:00"})
augmented_df = pd.concat([df, pd.DataFrame(synthetic_texts)])
augmented_df.to_csv("augmented_data.csv", index=False)
```

**Option 2: OpenAI GPT-4o:**

```
import pandas as pd
from openai import OpenAI

client = OpenAI(api_key="your-openai-api-key")
df = pd.read_csv("data.csv")
synthetic_texts = []
for text in df["text"]:
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[{"role": "user", "content": f"Paraphrase this: '{text}'"}],
        max_tokens=50
    )
    synthetic_texts.append({"id": len(df) + len(synthetic_texts) + 1, "text":
response.choices[0].message.content, "timestamp": "2025-03-27T16:00:00"})
augmented_df = pd.concat([df, pd.DataFrame(synthetic_texts)])
augmented_df.to_csv("augmented_data.csv", index=False)
```

**Option 3: LangChain with Claude:**

```
import pandas as pd
from langchain_anthropic import ChatAnthropic

model = ChatAnthropic(model="claude-3-sonnet-20240229",
```

```
api_key="your-anthropic-api-key")
df = pd.read_csv("data.csv")
synthetic_texts = []
for text in df["text"]:
    response = model.invoke(f"Paraphrase this text: '{text}'").content
    synthetic_texts.append({"id": len(df) + len(synthetic_texts) + 1, "text":
response, "timestamp": "2025-03-27T16:00:00"})
augmented_df = pd.concat([df, pd.DataFrame(synthetic_texts)])
augmented_df.to_csv("augmented_data.csv", index=False)
```

Example Output (augmented_data.csv):

```
id,text,timestamp
4,"Require immediate assistance with shipping delays","2025-03-27T16:00:00"
5,"Item quality is superb, speedy delivery","2025-03-27T16:00:00"
6,"Price is shockingly elevated for this product!!!","2025-03-27T16:00:00"
```

3. **Store Metadata in JSON:**

Example Metadata Script:

```
import json
metadata = {"original_rows": len(df), "augmented_rows": len(augmented_df),
"augmentation_ratio": len(augmented_df) / len(df)}
with open("metadata.json", "w") as f:
    json.dump(metadata, f)
```

**Expected Output:** augmented_data.csv and metadata.json.

**Alternate for Speed:** Use pre-written paraphrases instead of LLM calls (~20 min).

**Step 2: NLP Feature Extraction with LLMs (2 hr 30 min)**
**Objective:** Extract entities, sentiment, and themes using LLMs and frameworks.

**Tasks:**

1. **Extract Features:**

**Option 1: Gemini 2.0 Flash (Vertex AI:**

```
import pandas as pd
from vertexai.preview.language_models import TextGenerationModel

df = pd.read_csv("augmented_data.csv")
model = TextGenerationModel.from_pretrained("gemini-2.0-flash")

def extract_features(text):
    entity_prompt = f"List key entities in this text: '{text}'"
    entities = model.predict(entity_prompt, max_output_tokens=20).text.split(", ")
    sentiment_prompt = f"Determine the sentiment (positive, negative, neutral) of
this text: '{text}'"
    sentiment = model.predict(sentiment_prompt, max_output_tokens=10).text
    theme_prompt = f"Identify the main theme (e.g., urgency, quality, pricing) of
this text: '{text}'"
    theme = model.predict(theme_prompt, max_output_tokens=10).text
    return {"entities": entities, "sentiment": sentiment, "theme": theme}

features = df["text"].apply(extract_features)
df_features = pd.DataFrame(features.tolist())
df = pd.concat([df, df_features], axis=1)
df.to_csv("features_data.csv", index=False)
```

**Option 2: OpenAI GPT-4o:**

```
from openai import OpenAI

client = OpenAI(api_key="your-openai-api-key")
df = pd.read_csv("augmented_data.csv")

def extract_features(text):
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[
            {"role": "user", "content": f"Extract entities, sentiment
(positive/negative/neutral), and theme (e.g., urgency, quality, pricing) from:
'{text}'"}
        ],
        max_tokens=50
    )
    result = eval(response.choices[0].message.content)  # Assuming structured
output
    return result

features = df["text"].apply(extract_features)
```

```
df_features = pd.DataFrame(features.tolist())
df = pd.concat([df, df_features], axis=1)
df.to_csv("features_data.csv", index=False)
```

- 

**Option 3: LangChain with Multiple LLMs:**

```
from langchain_openai import ChatOpenAI
from langchain_anthropic import ChatAnthropic
from langchain.prompts import PromptTemplate
import pandas as pd

df = pd.read_csv("augmented_data.csv")
llm = ChatOpenAI(model="gpt-4o", api_key="your-openai-api-key")  # Or
ChatAnthropic("claude-3-sonnet-20240229")
prompt = PromptTemplate(
    input_variables=["text"],
    template="Extract entities, sentiment (positive/negative/neutral), and theme
(e.g., urgency, quality, pricing) from: '{text}'"
)

def extract_features(text):
    response = llm.invoke(prompt.format(text=text)).content
    result = eval(response)  # Assuming structured output
    return result

features = df["text"].apply(extract_features)
df_features = pd.DataFrame(features.tolist())
df = pd.concat([df, df_features], axis=1)
df.to_csv("features_data.csv", index=False)
```

Example Output (features_data.csv):

```
id,text,timestamp,entities,sentiment,theme
1,"Need urgent help...","2025-03-27T10:00:00",["delivery"],negative,urgency
2,"Product quality...","2025-03-27T12:00:00",["quality",
"shipping"],positive,quality
3,"Unusually high...","2025-03-27T14:00:00",["price"],negative,pricing
```

2. **Vectorize Text:**

**Option 1: Vertex AI Embeddings:**

```
from vertexai.preview.language_models import TextEmbeddingModel
embedding_model = TextEmbeddingModel.from_pretrained("text-embedding-001")
embeddings = [embedding_model.get_embeddings([text])[0].values for text in
df["text"]]
df["embedding"] = embeddings
```

**Option 2: OpenAI Embeddings:**

```
from openai import OpenAI
client = OpenAI(api_key="your-openai-api-key")
embeddings = [client.embeddings.create(model="text-embedding-3-small",
input=text).data[0].embedding for text in df["text"]]
df["embedding"] = embeddings
```

**Option 3: LangChain with Hugging Face:**

```
from langchain_huggingface import HuggingFaceEmbeddings
embedding_model =
HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
embeddings = [embedding_model.embed_query(text) for text in df["text"]]
df["embedding"] = embeddings
```

- Save: df.to_csv("vectorized_data.csv", index=False)
- **Expected Output:** vectorized_data.csv with features and embeddings.
- **Alternate for Speed:** Use TF-IDF from scikit-learn instead of embeddings (~30 min saved).

**Step 3: Anomaly Detection with ML (2 hr)**

**Objective:** Detect anomalies using embeddings and ML.

**Tasks:**

1. **Train Isolation Forest:**

Example Script:

```
from sklearn.ensemble import IsolationForest
import pandas as pd
import numpy as np
```

```
df = pd.read_csv("vectorized_data.csv")
X = np.array(df["embedding"].tolist())
model = IsolationForest(contamination=0.1, random_state=42)
model.fit(X)
df["anomaly_score"] = model.decision_function(X)
df["is_anomaly"] = model.predict(X) == -1
df.to_csv("anomaly_data.csv", index=False)
```

2. **Summarize Anomalies:**

**Option 1: Gemini 2.0 Flash:**

```
anomalies = df[df["is_anomaly"]]
for idx, row in anomalies.iterrows():
    prompt = f"Summarize why this text might be an anomaly: '{row['text']}'"
    summary = model.predict(prompt, max_output_tokens=50).text
    print(f"ID {row['id']}: {summary}")
```

**Option 2: OpenAI GPT-4o:**

```
for idx, row in anomalies.iterrows():
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[{"role": "user", "content": f"Summarize why this text might be an
anomaly: '{row['text']}'"}],
        max_tokens=50
    )
    print(f"ID {row['id']}: {response.choices[0].message.content}")
```

**Option 3: LangChain with Claude:**

```
model = ChatAnthropic(model="claude-3-sonnet-20240229",
api_key="your-anthropic-api-key")
for idx, row in anomalies.iterrows():
    summary = model.invoke(f"Summarize why this text might be an anomaly:
'{row['text']}'").content
    print(f"ID {row['id']}: {summary}")
Example Output:
```

```
ID 3: High price complaint is unusual due to strong negativity.
```

**Expected Output:** anomaly_data.csv and anomaly summaries.

**Alternate for Speed:** Skip summarization (~20 min saved).

#### Step 4: Validation & Experimentation (1 hr)

**Objective:** Validate and experiment with ML settings.
**Tasks:**

1. **Evaluate Performance:**

Example Script:

```
true_anomalies = [3]  # Hypothetical ground truth
predicted_anomalies = df[df["is_anomaly"]]["id"].tolist()
precision = len(set(true_anomalies) & set(predicted_anomalies)) /
len(predicted_anomalies)
recall = len(set(true_anomalies) & set(predicted_anomalies)) / len(true_anomalies)
print(f"Precision: {precision}, Recall: {recall}")
```

2. **Experiment with Contamination:**

Example Script:

```
import json
results = {}
for contam in [0.05, 0.15]:
    model = IsolationForest(contamination=contam)
    model.fit(X)
    results[contam] = sum(model.predict(X) == -1)
with open("metadata.json", "r+") as f:
    metadata = json.load(f)
    metadata["experiments"] = results
    f.seek(0)
    json.dump(metadata, f)
```

**Expected Output:** Updated metadata.json with metrics.

#### Step 5: AI Tool Integration (Throughout, 30 min total)

- **OpenAI ChatGPT, Claude, Gemin:** Generate prompts, debug scripts, suggest NLP approaches.

- **LLM Models:** Use for chat-based feature extraction or summarization (e.g., GPT-4o, Claude 3, Gemini 2.0 Flash).
- **Alternate for Speed:** Stick to Grok (~10 min saved).

**Deliverables**

1. **Functional Pipeline:**
    - Local Python scripts for augmentation, extraction, and detection.
2. **GitHub Repository:**
    - Code, CSV files, metadata.json.
    - README.md with:
        - Setup (e.g., pip install openai langchain-anthropic scikit-learn).
        - Usage (e.g., python augment.py).
        - LLM options and alternates.
3. **Report (1–2 Pages, PDF/Markdown):**
    - **Overview:** LLM and ML design.
    - **Implementation:**
        - Augmentation (Gemini/OpenAI/LangChain).
        - Extraction (Vertex/OpenAI/LangChain).
        - Detection (Isolation Forest).
    - **Metrics:**
        - Precision/recall (>70%).
        - Feature coverage (>90%).
    - **AI Impact:** Grok and LLM assistance.
    - **Challenges:** E.g., “API rate limits” mitigated with batching.

**Rubric**

- **ML Design & NLP Integration (30%)**
- **Anomaly Detection (25%)**
- **Data Augmentation (20%)**
- **Experimentation & Validation (15%)**
- **Documentation (10%)**

**Expanded Features & Considerations**

- **Flexibility:** Multiple LLM options (OpenAI, Claude, Gemini) via LangChain.
- **Scalability:** Local setup can transition to cloud with minimal changes.
- **Privacy:** Synthetic data protects original text.