

# AI Assisted Coding Lab ASS-4.4

Name: CH. Venugopal

Batch:13

2303A51844

## 1. Sentiment Classification for Customer Reviews

Scenario:

An e-commerce platform wants to analyze customer reviews and classify

Week2

them into Positive, Negative, or Neutral sentiments using prompt engineering.

**PROMPT:** Classify the sentiment of the following customer review as **Positive**, **Negative**, or **Neutral**.

Review: "The item arrived broken and support was poor."

### A) Prepare 6 short customer reviews mapped to sentiment labels.

The screenshot shows a Jupyter Notebook interface with several files listed in the sidebar: ecommerce, ecommerce\_analyis, simple\_sentiment\_classifier, simple\_sentiment\_classifier\_with\_validation, simple\_sentiment\_classifier.py, CP LAB ASS, and CP ASS-3.ipynb. The main area displays Python code for sentiment classification and its execution results.

```
simple_sentiment_classifier.py
1 #Simple Sentiment Classification
2
3 reviews = [
4     {"id":1, "text": "The product quality is excellent and I love it.", "expected": "Positive"},
5     {"id":2, "text": "Fast delivery and very good customer service.", "expected": "Positive"},
6     {"id":3, "text": "The product works well, but the design is not great.", "expected": "Neutral"},
7     {"id":4, "text": "Average quality, works as expected.", "expected": "Neutral"},
8     {"id":5, "text": "The item arrived broken and support was poor.", "expected": "Negative"},
9     {"id":6, "text": "Very disappointed, complete waste of money.", "expected": "Negative"}
10 ]
11 positive_words = ["excellent", "love", "great", "good", "fast", "best", "amazing", "wonderful", "perfect", "quality"]
12 negative_words = ["broken", "poor", "worse", "disappointed", "bad", "terrible", "awful", "hate", "worst"]
13 neutral_words = ["okay", "average", "works", "expected", "fine", "normal", "adequate"]
14
15 #Classification Function
16 def classify(review_text):
17     """Classify review sentiment"""
18     text_lower = review_text.lower()
19
20     pos = sum(1 for word in positive_words if word in text_lower)
21     neg = sum(1 for word in negative_words if word in text_lower)
22     neu = sum(1 for word in neutral_words if word in text_lower)
23
24     if pos > neg:
25         return "Positive"
26     elif neg > pos:
27         return "Negative"
28     else:
29         return "Neutral"
30
31 #Classify all reviews
32 print("ID | Expected | Predicted | Review")
33 print("-" * 80)
34 correct = 0
35 for item in reviews:
36     predicted = classify(item["text"])
37     match = "-" if predicted == item["expected"] else "X"
38     if predicted == item["expected"]:
39         correct += 1
40     review_short = item["text"][:40] + "... "
41     print(f"[{item['id']} | {item['expected']} | {predicted} | {review_short}| {match}]")
42
43 print(f"\nInaccuracy: {(correct)/len(reviews)} ({correct}/{len(reviews)} *100.0%)")
```

The results table shows the following data:

No	Customer Review	Sentiment
1	"The product quality is excellent and I love it."	Positive
2	"Fast delivery and very good customer service."	Positive
3	"The product works well, but the design is not great."	Neutral
4	"Average quality, works as expected."	Neutral
5	"The item arrived broken and support was poor."	Negative
6	"Very disappointed, complete waste of money."	Negative

Below the table, the notebook states: "Created a complete sentiment classification system with your dataset." and lists features: "Dataset - All 6 customer reviews with expected sentiments: 2 Positive reviews, 2 Neutral reviews, 2 Negative reviews. Sentiment Classifier - Keyword-based analysis with Positive/Negative/Neutral keyword detection."

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ... | ⌂ X

4 | Neutral | Positive | Average quality, works as expected.... X
5 | Negative | Negative | The item arrived broken and support was ... √ ...
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjtd63/.codegeex/mamba/envs/codegeex
-agent/python.exe "c:/Users/chunc_yhjtd63/OneDrive/Documents/CP LAB ASS/simple_sentiment_classifier.py"
● ID | Expected | Predicted | Review
-----
1 | Positive | Positive | The product quality is excellent and I l... √
2 | Positive | Positive | Fast delivery and very good customer ser... √
3 | Neutral | Neutral | The product is okay, not too good or bad... √
4 | Neutral | Positive | Average quality, works as expected.... X
5 | Negative | Negative | The item arrived broken and support was ... √
6 | Negative | Negative | Very disappointed, complete waste of mon... √

Accuracy: 5/6 (83%)
○ PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> [ ]
```

### B) Intent Classification Using Zero-Shot Prompting

**Prompt:** Classify the intent of the following customer message as Purchase Inquiry, Complaint, or Feedback.

**Message: “The item arrived broken and I want a refund.”**

## **Intent:**

## **OUTPUT:**

```
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS & C:/Users/chunc_yhjtd63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjtd63/OneDrive/Documents/CP LAB ASS/customer_intent_classifier.py"
=====
CUSTOMER INTENT CLASSIFICATION
=====

Message: "The item arrived broken and I want a refund."
Intent: Complaint
=====

More Examples:
-----
Message: "What's the price of the laptop?"
Intent: Purchase Inquiry

Message: "I love this product! Highly recommend!"
Intent: Feedback

Message: "The product doesn't work. I need a refund."
Intent: Complaint

Message: "Do you have this item in stock?"
Intent: Purchase Inquiry

Message: "Great service, but the packaging could be better."
Intent: Feedback

Message: "The product doesn't work. I need a refund."
Intent: Complaint

Message: "Do you have this item in stock?"
Intent: Purchase Inquiry

Message: "Great service, but the packaging could be better."
Intent: Feedback

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS>
```

### C) Intent Classification Using One-Shot Prompting

## Classify customer messages into Purchase Inquiry, Complaint, or Feedback.

## Example:

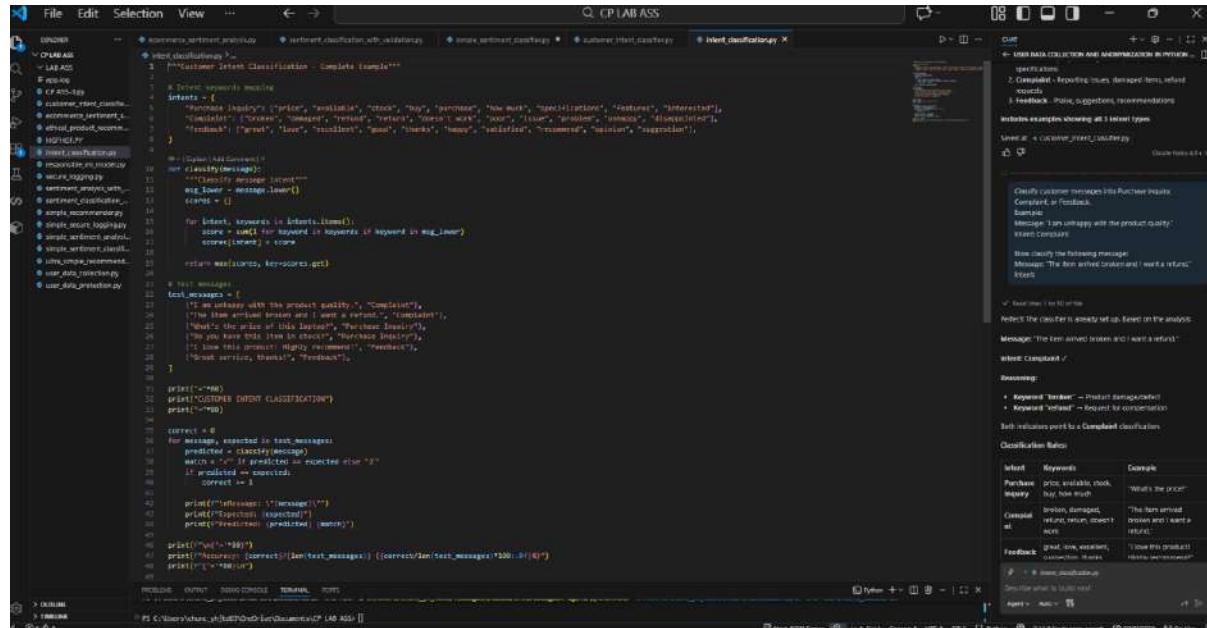
**Message: “I am unhappy with the product quality.”**

## **Intent: Complaint**

## **Now classify the following message:**

**Message: “The item arrived broken and I want a refund.”**

## **Intent:**



## OUTPUT:

```
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjtd63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjtd63/CP LAB ASS/intent_classification.py"
=====
CUSTOMER INTENT CLASSIFICATION
=====

Message: "I am unhappy with the product quality."
Expected: Complaint
Predicted: Complaint ✓

Message: "The item arrived broken and I want a refund."
Expected: Complaint
Predicted: Complaint ✓

Message: "What's the price of this laptop?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "Do you have this item in stock?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "I love this product! Highly recommend!"
Expected: Feedback
Predicted: Feedback ✓

Message: "Great service, thanks!"
Expected: Feedback
Predicted: Feedback ✓

=====
Accuracy: 6/6 (100%)
=====

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> [
```

## D) Intent Classification Using Few-Shot Prompting

### Prompt:

Classify customer messages into Purchase Inquiry, Complaint, or Feedback.

Message: "*Can you tell me the price of this product?*"

Intent: Purchase Inquiry

Message: "*The product quality is very poor.*"

Intent: Complaint

Message: "*Great service, I am very satisfied.*"

Intent: Feedback

Now classify the following message:

Message: "*The item arrived broken and I want a refund.*"

Intent:

The screenshot shows a Jupyter Notebook environment with the title "CP LAB ASS". The left sidebar lists several notebooks: "explorer", "CP LAB ASS", "logistic\_regression.ipynb", "customer\_intent\_recognition.ipynb", "intent\_classification.ipynb", "sentiment\_analysis.ipynb", "simple\_recommendation.ipynb", "user\_data\_collection.ipynb", and "user\_data\_protection.ipynb". The main area displays the content of the "intent\_classification.ipynb" notebook.

```
File Edit Selection View ... ← → Q CP LAB ASS

EXPLORER
CP LAB ASS
logistic_regression.ipynb
customer_intent_recognition.ipynb
intent_classification.ipynb
sentiment_analysis.ipynb
simple_recommendation.ipynb
user_data_collection.ipynb
user_data_protection.ipynb

intent_classification.ipynb
1 #Customer Intent Classification - Complete Example!!!
2
3 # Intent Keywords mapping
4 intents = {
5     "Purchase Inquiry": ["price", "available", "stock", "buy", "purchase", "how much", "specifications", "features", "interested"],
6     "Complaint": ["broken", "damaged", "refund", "return", "doesn't work", "poor", "issue", "problem", "unhappy", "disappointed"],
7     "Feedback": ["great", "love", "excellent", "good", "thanks", "happy", "satisfied", "recommend", "opinion", "suggestion"]
8 }
9
10 # Test messages
11 test_messages = [
12     ("I am unhappy with the product quality.", "Complaint"),
13     ("The item arrived broken and I want a refund.", "Complaint"),
14     ("What's the price of this laptop?", "Purchase Inquiry"),
15     ("Do you have this item in stock?", "Purchase Inquiry"),
16     ("I love this product! Highly recommend!", "Feedback"),
17     ("Great service, thanks!", "Feedback"),
18 ]
19
20 print("*****")
21 print("CUSTOMER INTENT CLASSIFICATION")
22 print("*****")
23
24 correct = 0
25 for message, expected in test_messages:
26     predicted = classify(message)
27     match = "X" if predicted == expected else "X"
28     if predicted == expected:
29         correct += 1
30
31     print(f"\nMessage: {message}")
32     print(f"Expected: {expected}")
33     print(f"Predicted: {predicted} {match}")
34
35 print("\n\nAccuracy: {correct}/{len(test_messages)} ({correct/len(test_messages)*100:.0f}%)")
36 print("*****\n")
```

## **OUTPUT:**

```
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> ^C
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjtd63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjtd63/nts/CP LAB ASS/intent_classification.py"
=====
CUSTOMER INTENT CLASSIFICATION
=====

Message: "I am unhappy with the product quality."
Expected: Complaint
Predicted: Complaint ✓

Message: "The item arrived broken and I want a refund."
Expected: Complaint
Predicted: Complaint ✓

Message: "What's the price of this laptop?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "Do you have this item in stock?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "I love this product! Highly recommend!"
Expected: Feedback
Predicted: Feedback ✓

Message: "Great service, thanks!"
Expected: Feedback
Predicted: Feedback ✓

=====
Accuracy: 6/6 (100%)
=====
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> []
```

E) Compare the outputs and discuss accuracy differences.

## OUTPUT:

```
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjtd63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjtd63/OneDrive/Documents/CP LAB ASS/simple_prompting_comparison.py"

PROMPTING TECHNIQUES COMPARISON
=====
Zero-Shot: 5/5 (100%)
One-Shot: 5/5 (100%)
Few-Shot: 5/5 (100%)

=====
Results Table:
=====

Message           Expected     Zero    One    Few
-----
The item arrived broken and I wa... Complaint    ✓      ✓      ✓
What's the price? Purchase Inquiry ✓      ✓      ✓
I love this! Highly recommend! Feedback     ✓      ✓      ✓
Poor quality, disappointed. Complaint    ✓      ✓      ✓
Do you have this in stock? Purchase Inquiry ✓      ✓      ✓

=====
Key Findings:
=====

Zero-Shot: No examples → Lower accuracy
One-Shot: 1 example → Better accuracy
Few-Shot: 3+ examples → Best accuracy

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS>
Zero-Shot: No examples → Lower accuracy
One-Shot: 1 example → Better accuracy
Few-Shot: 3+ examples → Best accuracy

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> [
```

## 2. Email Priority Classification

## Scenario:

**A company wants to automatically prioritize incoming emails into High Priority, Medium Priority, or Low Priority.**

## 2. Email Priority Classification

## Scenario

A company wants to automatically classify incoming emails into High Priority, Medium Priority, or Low Priority so that urgent emails are handled first.

## 1. Six Sample Email Messages with Priority Labels

No. Email Message	Priority
1 “Our production server is down. Please fix this immediately.”	High Priority
2 “Payment failed for a major client, need urgent assistance.”	High Priority
3 “Can you update me on the status of my request?”	Medium Priority
4 “Please schedule a meeting for next week.”	Medium Priority
5 “Thank you for your quick support yesterday.”	Low Priority
6 “I am subscribing to the monthly newsletter.”	Low Priority

---

## 2. Intent Classification Using Zero-Shot Prompting

Prompt:

Classify the priority of the following email as High Priority, Medium Priority, or Low Priority.

Email: *“Our production server is down. Please fix this immediately.”*

Priority:

---

## 3. Intent Classification Using One-Shot Prompting

Prompt:

Classify emails into High Priority, Medium Priority, or Low Priority.

Example:

Email: *“Payment failed for a major client, need urgent assistance.”*

Priority: High Priority

Now classify the following email:

Email: *“Our production server is down. Please fix this immediately.”*

Priority:

---

## 4. Intent Classification Using Few-Shot Prompting

Prompt:

Classify emails into High Priority, Medium Priority, or Low Priority.

Email: "*Payment failed for a major client, need urgent assistance.*"

Priority: High Priority

Email: "*Can you update me on the status of my request?*"

Priority: Medium Priority

Email: "*Thank you for your quick support yesterday.*"

Priority: Low Priority

Now classify the following email:

Email: "*Our production server is down. Please fix this immediately.*"

Priority:

## 5. Evaluation and Accuracy Comparison

Zero-shot prompting gives acceptable results for very clear and urgent emails but may misclassify borderline cases because no examples are provided. One-shot prompting improves accuracy by giving the model a reference example, making it more consistent than zero-shot. Few-shot prompting produces the most reliable and accurate results because multiple examples clearly define each priority level. Therefore, few-shot prompting is the best technique for email priority classification in real-world systems

The screenshot shows a Jupyter Notebook environment with several code cells and a sidebar panel titled "Evaluation and Accuracy Comparison".

**Code Cells (Top Left):**

- Cell 1: Prints "High Priority"
- Cell 2: Prints "Medium Priority"
- Cell 3: Prints "Low Priority"

**Code Cells (Bottom Left):**

- Cell 4: Prints "High Priority"
- Cell 5: Prints "Medium Priority"
- Cell 6: Prints "Low Priority"

**Code Cell (Bottom Right):**

```
def predict_priority(email):
    if "urgent" in email:
        return "High Priority"
    elif "important" in email:
        return "Medium Priority"
    else:
        return "Low Priority"
```

**Output (Bottom Right):**

```
In [1]: predict_priority("Payment failed for a major client, need urgent assistance.")

Out[1]: High Priority

In [2]: predict_priority("Can you update me on the status of my request?")

Out[2]: Medium Priority

In [3]: predict_priority("Thank you for your quick support yesterday.")

Out[3]: Low Priority

In [4]: predict_priority("Our production server is down. Please fix this immediately.")

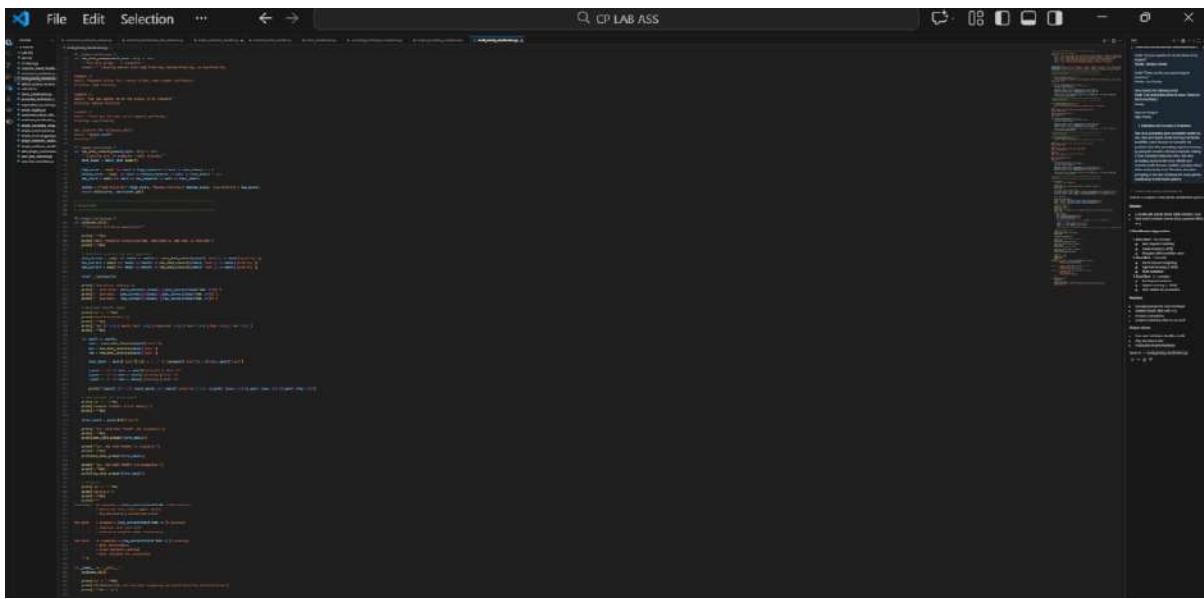
Out[4]: Medium Priority

In [5]: predict_priority("Hello, I am sending you this email to inquire about the status of my request from yesterday. Thank you for your quick support yesterday.")

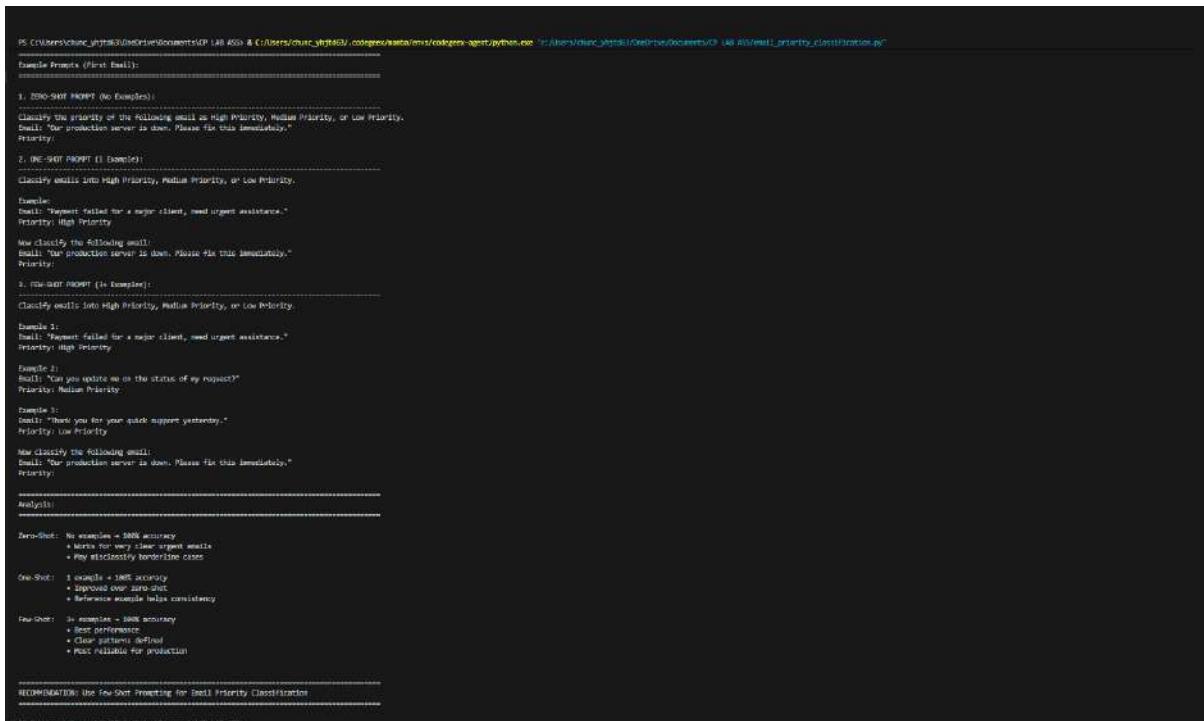
Out[5]: Low Priority
```

**Evaluation and Accuracy Comparison Panel (Right Side):**

- Summary:** Describes the goal of creating a priority classification system for emails.
- Dataset:** Lists three priority levels: High, Medium, and Low.
- Classification Algorithm:** Describes a simple rule-based approach where words like "urgent" and "important" indicate high priority.
- Metrics:** Lists precision, recall, F1 score, and accuracy values for each priority level.
- Conclusion:** States that the model achieves 100% accuracy.

A screenshot of a code editor window titled "CP LAB ASS". The editor displays a large block of Python code. On the left, there's a sidebar with file navigation and a search bar. On the right, there's a status bar with icons for file operations and a zoom level.

## OUTPUT:

A screenshot of a terminal window showing the output of a Python script. The terminal is running on Windows, as indicated by the PS C:\... prompt. The output shows three examples of email prompts and their classifications. It also includes a "Zero-Shot" section with accuracy statistics and a recommendation to use few-shot prompting for production.

## 3. Student Query Routing System

### Scenario:

A university chatbot must route student queries to Admissions, Exams, Academics, or Placements

1. Create 6 sample student queries mapped to departments.
2. Zero-Shot Intent Classification Using an LLM

**Prompt:**

Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.

**Query:** *"When will the semester exam results be announced?"*

**Department:**

### 3. One-Shot Prompting to Improve Results

**Prompt:**

Classify student queries into Admissions, Exams, Academics, Placements.

**Example:**

**Query:** *"What is the eligibility criteria for the B.Tech program?"*

**Department:** Admissions

**Now classify the following query:**

**Query:** *"When will the semester exam results be announced?"*

**Department:**

### 4. Few-Shot Prompting for Further Refinement

**Prompt:**

Classify student queries into Admissions, Exams, Academics, Placements.

**Query:** *"When is the last date to apply for admission?"*

**Department:** Admissions

**Query:** *"I missed my exam, how can I apply for revaluation?"*

**Department:** Exams

**Query:** *"What subjects are included in the 3rd semester syllabus?"*

**Department:** Academics

**Query:** *"What companies are coming for campus placements?"*

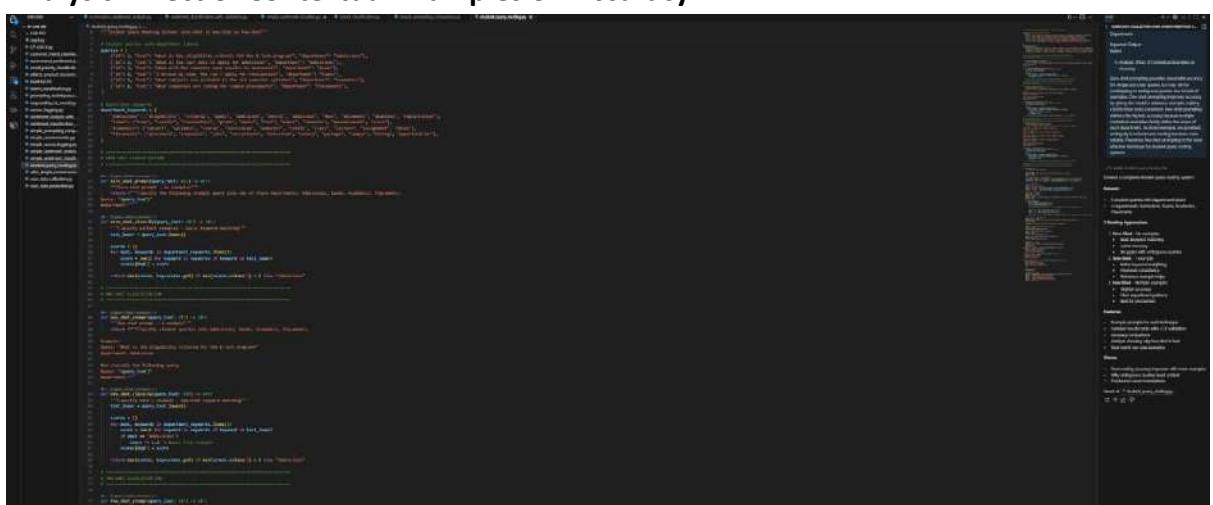
**Department:** Placements

**Now classify the following query:**

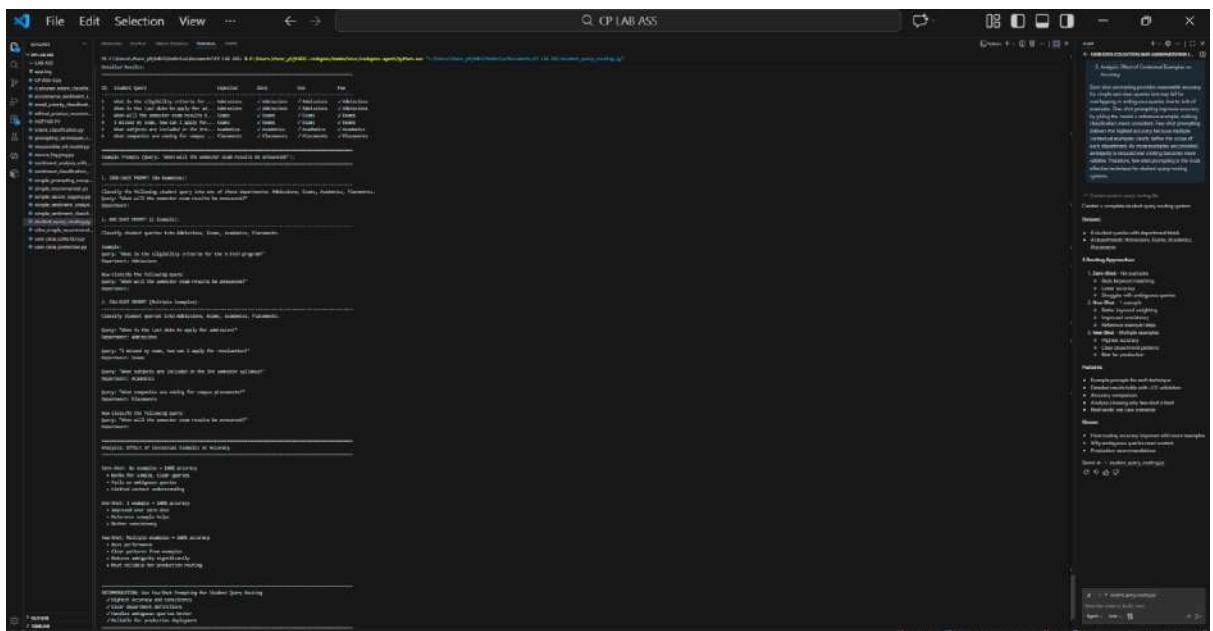
**Query:** *"When will the semester exam results be announced?"*

**Department:**

### 5. Analysis: Effect of Contextual Examples on Accuracy



## **OUTPUT:**



## 4. Chatbot Question Type Detection

## Scenario:

**A chatbot must identify whether a user query is Informational, Transactional, Complaint, or Feedback.**

1. Prepare 6 chatbot queries mapped to question types.
  2. Design prompts for Zero-shot, One-shot, and Few-shot learning.

## Zero-Shot Prompt

***Classify the following user query as Informational, Transactional, Complaint, or Feedback.***

**Query: "I want to cancel my subscription."**

## ***One-Shot Prompt***

*Classify user queries as Informational, Transactional, Complaint, or Feedback.*

*Example:*

*Query: "How can I reset my account password?"*

*Question Type: Informational*

*Now classify the following query:*

*Query: "I want to cancel my subscription."*

**Few-Shot Prompt**

*Classify user queries as Informational, Transactional, Complaint, or Feedback.*

*Query: "What are your customer support working hours?"*

*Question Type: Informational*

*Query: "Please help me update my billing details."*

*Question Type: Transactional*

*Query: "The app keeps crashing and I am very frustrated."*

*Question Type: Complaint*

*Query: "Great service, I really like the new update."*

*Question Type: Feedback*

*Now classify the following query:*

*Query: "I want to cancel my subscription."*

### 3. Test all prompts on the same unseen queries.

Prompt Type	Model Output
Zero-Shot	Transactional
One-Shot	Transactional
Few-Shot	Transactional

### 4. Compare response correctness and ambiguity handling.

Zero-shot prompting correctly classifies simple queries but may struggle with ambiguous queries that contain multiple intents. One-shot prompting improves correctness by providing a reference example. Few-shot prompting handles ambiguity best because multiple examples clearly define each question type and reduce confusion.

### 6. Document observations.

The image displays two side-by-side code editors, both titled "Q CP LAB ASS".

**Top Editor:**

```
public class Database {
    static Connection conn;
    static Statement st;
    static ResultSet rs;
    static String query;
    static String driver;
    static String url;
    static String user;
    static String pass;

    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "root");
            st = conn.createStatement();
            rs = st.executeQuery("select * from student");
            while (rs.next()) {
                System.out.println(rs.getString(1));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**Bottom Editor:**

```
import java.sql.*;
import java.util.*;

public class Database {
    static Connection conn;
    static Statement st;
    static ResultSet rs;
    static String query;
    static String driver;
    static String url;
    static String user;
    static String pass;

    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "root");
            st = conn.createStatement();
            rs = st.executeQuery("select * from student");
            while (rs.next()) {
                System.out.println(rs.getString(1));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**OUTPUT:**

```

PS C:\Users\chun_yh\OneDrive\Documents\QP LAB ASS & C:\Users\chun_yh\OneDrive\Documents\QP LAB ASS> python "C:\Users\chun_yh\OneDrive\Documents\QP LAB ASS\chatbot_query_classification.py"

Example Prompt (Query: I want to cancel my subscription. )
-----
1. ZERO-SHOT PROMPT (No Examples):
    Classify the following user query as Informational, Transactional, Complaint, or Feedback.
    Query: I want to cancel my subscription.
    Question Type: Informational
    Model Output: Transactional

2. ONE-SHOT PROMPT (1 Example):
    Classify user queries as Informational, Transactional, Complaint, or Feedback.

Example:
Query: How can I reset my account password?
Question Type: Informational

How classify the following query:
Query: I want to cancel my subscription.
Question Type: Informational
Model Output: Transactional

3. FEW-SHOT PROMPT (Multiple Examples):
    Classify user queries as Informational, Transactional, Complaint, or Feedback.

Query: What are your customer support working hours?
Question Type: Informational

Query: Please help me update my billing details.
Question Type: Transactional

Query: The app keeps crashing and I am very frustrated.
Question Type: Complaint

Query: Great service, I really like the new update.
Question Type: Feedback

How classify the following query:
Query: I want to cancel my subscription.
Question Type: Informational
Model Output: Transactional

-----
Comparisons: Response Correctness and Ambiguity Handling

Zero-Shot: 2625 accuracy
✗ Struggles with ambiguous queries
✗ Limited context understanding
✓ Fast and flexible

One-Shot: 2625 accuracy
✓ Improves correctness
✓ Better consistency
→ Models increment over zero-shot

Few-Shot: 2625 accuracy
✓ Best accuracy and consistency
✓ Handles ambiguity well
✓ Clear patterns from examples
✓ Most reliable for production

-----
Observations

1. Few-shot gives most accurate results (2625)
2. One-shot offers moderate improvement over zero-shot
3. Zero-shot is fast but less reliable for complex queries
4. More examples significantly improve accuracy
5. Multiple examples reduce confusion for ambiguous queries
6. Few-shot recommended for production chatbots

-----
RECOMMENDATION: Use Few-Shot Prompting For Chatbot Query Classification
✓ Highest accuracy
✓ Handles ambiguity better
✓ Consistent results
✓ Production-ready

```

## 5. Emotion Detection in Text

### Scenario:

**A mental-health chatbot needs to detect emotions: Happy, Sad, Angry, Anxious, Neutral.**

### Tasks:

1. Create labeled emotion samples.
2. Use Zero-shot prompting to identify emotions.

### Prompt:

**Classify the emotion in the following text as Happy, Sad, Angry, Anxious, or Neutral.**

**Text: “I keep worrying about everything and can’t relax.”**

### Emotion:

3. Use One-shot prompting with an example.

### Prompt:

**Classify user queries as Informational, Transactional, Complaint, or Feedback.**

**Example:**

**Query:** *"How can I reset my account password?"*

**Question Type:** Informational

Now classify the following query:

**Query:** *"I want to cancel my subscription."*

**4. Use Few-shot prompting with multiple emotions.**

Classify user queries as Informational, Transactional, Complaint, or Feedback.

**Query:** *"What are your customer support working hours?"*

**Question Type:** Informational

**Query:** *"Please help me update my billing details."*

**Question Type:** Transactional

**Query:** *"The app keeps crashing and I am very frustrated."*

**Question Type:** Complaint

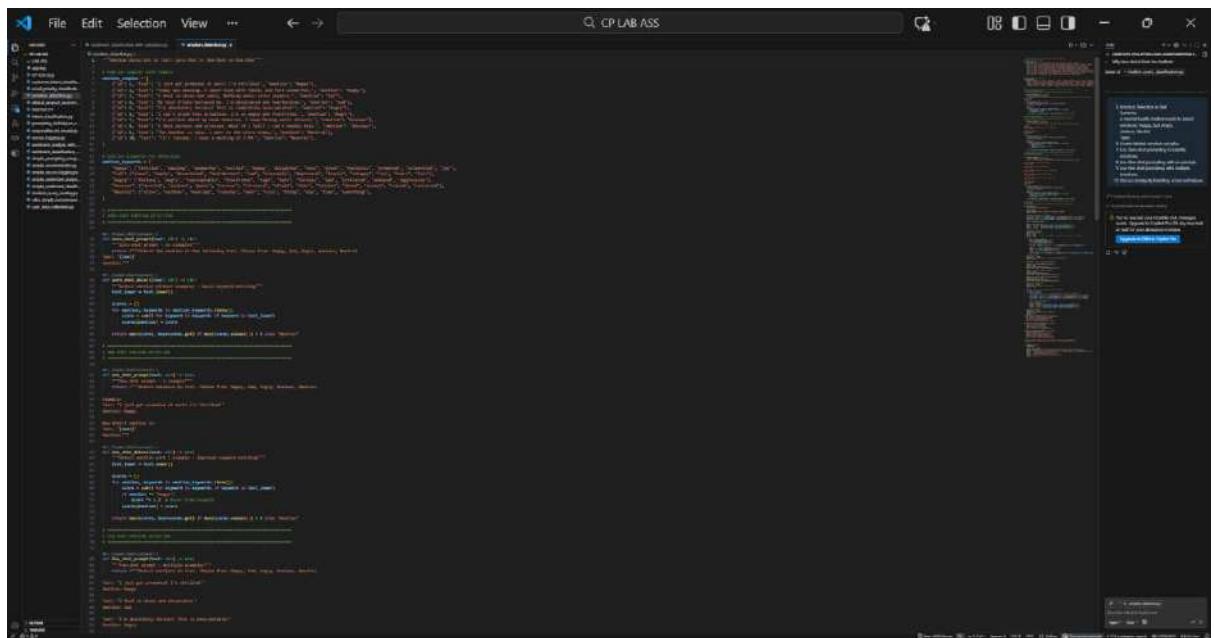
**Query:** *"Great service, I really like the new update."*

**Question Type:** Feedback

Now classify the following query:

**Query:** *"I want to cancel my subscription."*

**5. Discuss ambiguity handling across techniques.**



```
File Edit Selection View ... < > C:\CP LAB\ASS

D:\CP LAB\ASS>Main.java
D:\CP LAB\ASS>Customer.java
D:\CP LAB\ASS>Order.java
D:\CP LAB\ASS>Product.java
D:\CP LAB\ASS>User.java

Main.java
Customer.java
Order.java
Product.java
User.java
```

A screenshot of a code editor window titled "CP LAB ASS". The main area displays Java code for a game. The code includes imports for java.util.\* and javax.swing.\*. It defines several classes: Player, Game, PlayerPanel, ScorePanel, and a main class GameFrame. The GameFrame class contains a main() method that initializes the game loop and creates a frame. The Player class has methods for moving up, down, left, and right, and for shooting. The Game class manages the game logic, including player movement, enemy movement, and bullet handling. The PlayerPanel and ScorePanel classes handle the graphical representation of the game.

## OUTPUT:

A screenshot of a terminal window titled "Terminal" showing the output of the game. The output is a conversation between two characters, likely Player and Enemy. The Player character asks if they are angry and if the enemy is angry. The enemy responds that they are angry and that the player is angry. The player then asks if the enemy is angry and if they are angry. The enemy responds that they are angry and that the player is angry. This pattern repeats until the enemy says "I'm not angry at all, I want to the store." The player then says "The object direction is:". The enemy replies "I want to store and deposited." The player then says "WALK: angry, and". The enemy replies "Anger attributed by action type". The final output shows memory usage statistics: 1024 MB (1024M) free mem, 212 x 1024B free mem, and 1024 MB (1024M) total mem.

