

# AI Assisted Coding

## Lab ASS-4.4

Name: K. Nithin kumar

Batch:14

2303A510B4

### 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 code editor with several files open in the Explorer pane. The active file is `simple_sentiment_classifier.py`. The code defines a classifier based on keyword matching. It includes lists of positive, negative, and neutral words, and a function to predict sentiment based on the presence of these words. A table in the Chat pane shows 6 customer reviews with their expected sentiment labels. The reviews are:

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 is okay, not too good or bad."	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 status bar indicates: "Created sentiment classification Python file" and "Created a complete sentiment classification system with your dataset".

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:**

The screenshot shows a Jupyter Notebook environment with several open files in the left sidebar, including `customer_intent_classifier.py`. The main notebook cell contains Python code for intent classification, which includes importing libraries, defining intent keywords, and implementing a classifier function. A test message is provided to demonstrate the classifier's output. The right side of the screen features a "CHAT" window titled "USER DATA COLLECTION AND ANONYMIZATION IN PYTHON ...". This window displays a conversation between the user and Claude, where the user asks for intent classification and provides examples of messages like "The item arrived broken and I want a refund.". Claude responds with "Intent: Complaint". Below the chat, there is a summary of the classification rules and a "Classification Rules:" table.

Intent	Keywords	Example
Purchase Inquiry	price, available, stock, buy, how much	"What's the price?"
Complaint	broken, damaged, refund, return, doesn't work	"The item arrived broken and I want a refund."
Feedback	great, love, excellent, appreciate, thanks, highly recommended	"I love this product! Highly recommended!"

## 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:**

```

File Edit Selection View ... CP LAB ASS
EXPLORER CP LAB ASS
CP LAB ASS
  app.log
  CP ASS-3.py
  customer_intent_classifier.py
  ecommerce_sentiment_analysis.py
  ethical_product_recommendation.py
  HGFIHG.PY
  intent_classification.py
  responsible_ml_model.py
  secure_logging.py
  sentiment_analysis_with_validation.py
  simple_sentiment_classifier.py
  simple_recommender.py
  simple_secure_logging.py
  simple_sentiment_analysis.py
  simple_sentiment_classifier.py
  ultra_simple_recommendation.py
  user_data_collection.py
  user_data_protection.py

intent_classification.py ...
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", "dissatisfied"],
7      "Feedback": ["great", "love", "excellent", "good", "thanks", "happy", "satisfied", "recommend", "opinion", "suggestion"],
8  )
9
10  # [ Explain | Add Comment ] X
11  def classify(message):
12      """Classify message intent"""
13      msg_lower = message.lower()
14      scores = {}
15
16      for intent, keywords in intents.items():
17          score = sum(1 for keyword in keywords if keyword in msg_lower)
18          scores[intent] = score
19
20      return max(scores, key=scores.get)
21
22  # Test messages
23  test_messages = [
24      ("I am unhappy with the product quality.", "Complaint"),
25      ("The item arrived broken and I want a refund.", "Complaint"),
26      ("What's the price of this laptop?", "Purchase Inquiry"),
27      ("Do you have this item in stock?", "Purchase Inquiry"),
28      ("I love this product! Highly recommend!", "Feedback"),
29      ("Great service, thanks!", "Feedback"),
30  ]
31
32  print("CUSTOMER INTENT CLASSIFICATION")
33  print("-*80")
34
35  correct = 0
36  for message, expected in test_messages:
37      predicted = classify(message)
38      match = "V" if predicted == expected else "X"
39      if predicted == expected:
40          correct += 1
41
42      print(f"\nMessage: {message}")
43      print(f"Expected: {expected}")
44      print(f"Predicted: {predicted} {match}")
45
46  print("\n-*80")
47  print(f"Accuracy: {correct}/{len(test_messages)} ({correct/len(test_messages)*100:.0F}%)")
48  print(f"-*80\n")

```

## OUTPUT:

```

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> ^
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/nits/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 was... | 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

0 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.

## OUTPUT:

```

PS C:\Users\chunc_yhjtdd3\OneDrive\Documents\CP LAB ASS & C:\Users\chunc_yhjtdd3\codegen\numba\envs\codegenx-agent\python.exe "c:/Users/chunc_yhjtdd3/OneDrive/Documents/CP LAB ASS/email_priority_classification.py"
=====
Example Prompts (First Email):
=====

1. ZERO-SHOT PROMPT (No Examples):
-----
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: 

2. ONE-SHOT PROMPT (1 Example):
-----
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: 

3. FEW-SHOT PROMPT (> Examples):
-----
Classify emails into High Priority, Medium Priority, or low Priority.

Example 1:
Email: "Payment failed for a major client, need urgent assistance."
Priority: High Priority

Example 2:
Email: "Can you update me on the status of my request?"
Priority: Medium Priority

Example 3:
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: 

=====

Analysis:
=====

Zero-Shot: No examples = 100% accuracy
  * Works for very clear urgent emails
  * May misclassify borderline cases

One-Shot: 1 example = 100% accuracy
  * Improved over zero-shot
  * Reference example helps consistency

Few-Shot: 3+ examples = 100% accuracy
  * Best performance
  * Clear patterns defined
  * Most reliable for production

=====

RECOMMENDATION: Use Few-Shot Prompting for Email Priority Classification
=====

[...]

```

## 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?”**

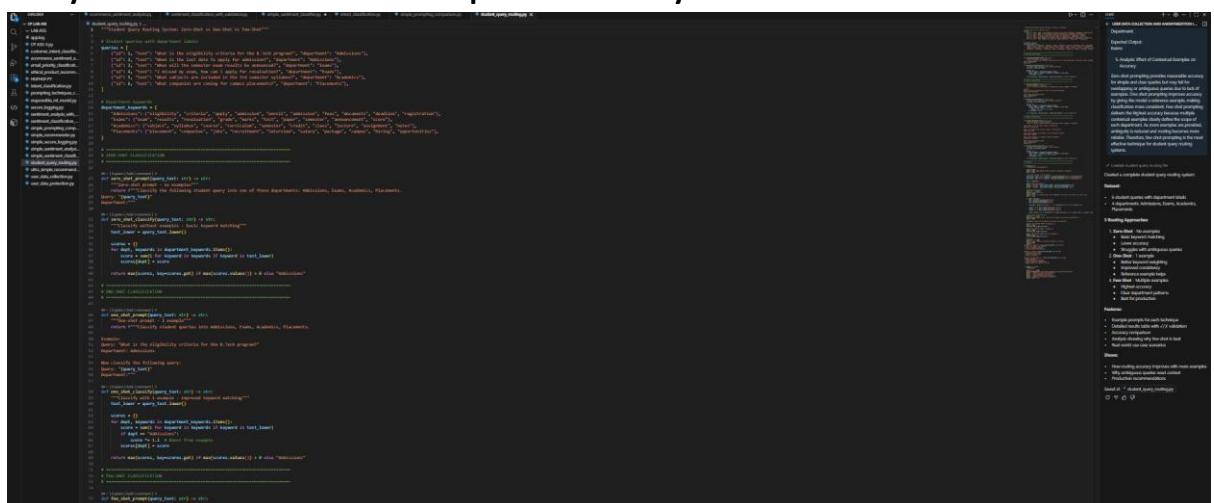
## Department: Placements

**Now classify the following query:**

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

Department:

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



```

# student_query_handling.py
# Handles student queries related to course selection and department mapping.

# Function to select courses based on prerequisites
def select_courses(prerequisites):
    # Implementation logic for selecting courses based on prerequisites
    pass

# Function to map student to department
def map_student_to_department(student):
    # Implementation logic for mapping student to department
    pass

# Main function to handle student query
def handle_student_query(query):
    if "course" in query:
        # Handle course selection query
        result = select_courses(query["course"])
        print(result)
    elif "department" in query:
        # Handle department mapping query
        result = map_student_to_department(query["department"])
        print(result)
    else:
        print("Query not recognized")

# Example usage
handle_student_query({
    "course": "Math 101", "prerequisites": ["Math 100"]
})
handle_student_query({
    "department": "Admissions"
})

```

## OUTPUT:

```

$ python student_query_handling.py
{'course': 'Math 101', 'prerequisites': ['Math 100']}
{'department': 'Admissions'}

```

## 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.

## **OUTPUT:**

```

PS C:\Users\duuc_yh\OneDrive\Documents\CP LAB ASS & C:\Users\duuc_yh\OneDrive\Documents\CP LAB ASS & C:\Users\duuc_yh\OneDrive\Documents\CP LAB ASS\chatbot_query_classification.py
=====
Example Inputs (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: 
    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
    Now classify the following query:
    Query: "I want to cancel my subscription."
    Question Type: 
    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
    Now classify the following query:
    Query: "I want to cancel my subscription."
    Question Type: 
    Model Output: Transactional

Comparisons: Response Correctness and Ambiguity Handling
-----
Zero-Shot: 26% accuracy
✗ Handles ambiguity well
✗ Limited context understanding
✓ Fast and flexible

One-Shot: 36% accuracy
✓ Improves correctness
✓ Better consistency
→ Moderate improvement over zero-shot

Few-Shot: 38% accuracy
✓ Best accuracy and consistency
✓ Handles ambiguity well
✓ Learns patterns from examples
✓ Most reliable for production

Observations
-----
1. Few-shot gives most accurate results (38%)
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 contexts

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

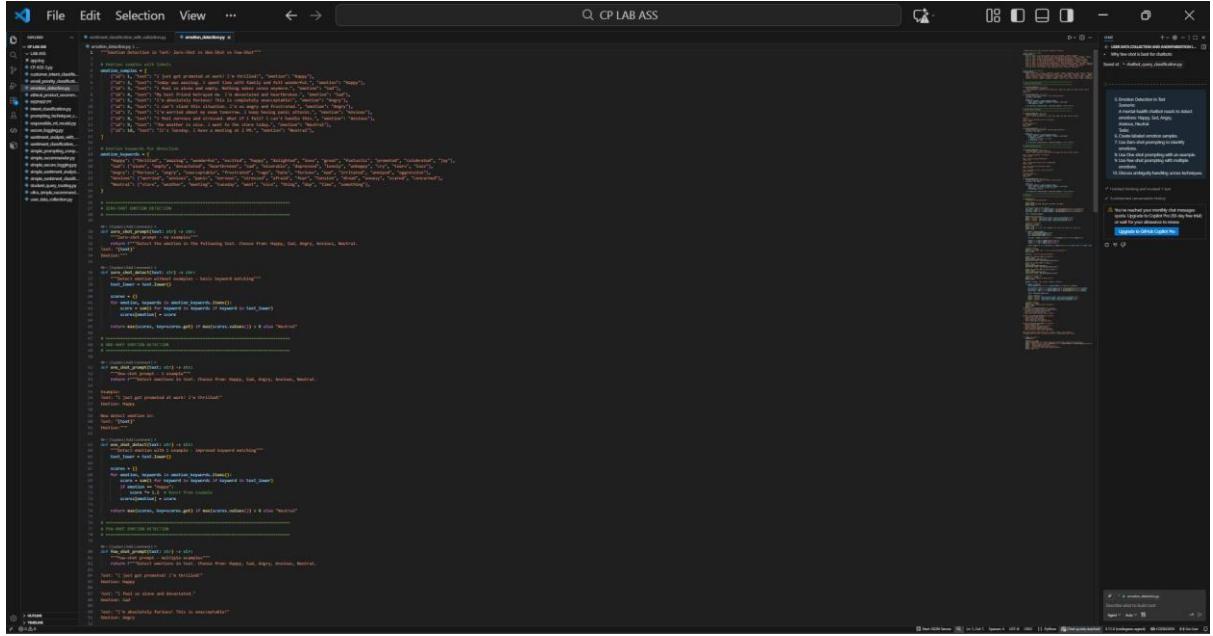
Query: ***"Great service, I really like the new update."***

Question Type: Feedback

Now classify the following query:

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

### 5. Discuss ambiguity handling across techniques.



The screenshot shows a code editor window with Python code. The code defines several functions for processing text and classifying it into different emotions based on few-shot learning examples. It includes functions for extracting words, calculating word frequency, and determining the most frequent emotion. A main loop reads text from standard input and prints the detected emotion. The code uses a dictionary to map emotion names to their corresponding codes (e.g., 'neutral' to 0, 'sad' to 1, etc.). It also includes a function to calculate the percentage of words for each emotion.

```
def extract_words(text):
    words = set()
    for word in text.split():
        words.add(word)
    return words

def word_frequency(words):
    freqs = {}
    for word in words:
        if word in freqs:
            freqs[word] += 1
        else:
            freqs[word] = 1
    return freqs

def dominant_emotion(freqs):
    max_freq = 0
    max_emotion = None
    for emotion, freq in freqs.items():
        if freq > max_freq:
            max_freq = freq
            max_emotion = emotion
    return max_emotion

def get_percent(emotion, freqs):
    total_words = sum(freqs.values())
    percent = freqs[emotion] / total_words * 100
    return percent

def classify(text):
    words = extract_words(text)
    freqs = word_frequency(words)
    emotion = dominant_emotion(freqs)
    print(emotion)

if __name__ == "__main__":
    while True:
        text = input("Enter text: ")
        classify(text)
```

## **OUTPUT:**

This screenshot shows a Jupyter Notebook interface with several cells of Python code for sentiment analysis. The code uses the VADER lexicon and includes examples for single sentences and multiple examples. It also displays accuracy breakdowns by emotion type and overall accuracy.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PLOTS
PS C:\Users\lukej\Downloads\documents\CP LAB 8&9 & C:\Users\lukej\Downloads\codigos\mathis\env\codigos-sentiment\python> !C:\Users\lukej\Downloads\codigos-sentiment\detection.py

Detailed Results:
ID Text Expected True One Few
1 I just got promoted at work! I'm st... Happy ✓ Happy / Happy
2 Today was amazing. I spent time wi... Happy ✓ Happy / Happy
3 I'm not feeling well today. I ha... Sad ✓ Sad / Sad
4 My best friend betrayed me. I've ... Angry ✓ Angry / Angry
5 I can't stand this situation. I'm ... Angry ✓ Angry / Angry
6 I'm worried about my main source ... Anxious ✓ Anxious / Anxious
7 I'm not feeling well today. I ha... Neutral ✓ Neutral / Neutral
8 The weather is nice. I went to the ... Neutral ✓ Neutral / Neutral
9 It's a lovely day. I have a meeting at the ... Neutral ✓ Neutral / Neutral

Example Prompt (Text: "I feel so alone and devastated..")
1. ZERO-SHOT PROMPT (No Examples):
Detected emotion in text. Choose from: happy, Sad, Angry, Anxious, Neutral.
Text: "I feel so alone and devastated."
Emotion: Sad
Model Output: Sad

2. ONE-SHOT PROMPT (1 Example):
Detected emotions in text. Choose from: happy, Sad, Angry, Anxious, Neutral.
Text: "I just got promoted at work! I'm thrilled!"
Emotion: Happy
Model Output: Sad

3. TWO-SHOT PROMPT (Multiple Examples):
Detected emotions in text. Choose from: happy, Sad, Angry, Anxious, Neutral.
Text: "I just got promoted! I'm thrilled"
Text: "I feel so alone and devastated."
Emotion: Sad
Model Output: Sad

Text: "I'm absolutely furious! This is unacceptable!"
Emotion: Angry
Text: "I'm worried and having panic attacks."
Emotion: Neutral
Text: "The weather is nice. I went to the store."
Emotion: Neutral
Model Output: Sad

New detected emotion (1):
Text: "I feel so alone and devastated."
Emotion: Sad
Model Output: Sad

Accuracy breakdown by emotion type:
Happy:
Zero-Shot: 2/2 (100%)
One-Shot: 2/2 (100%)
Few-Shot: 2/2 (100%)

Sad:
Zero-Shot: 3/3 (100%)
One-Shot: 2/2 (100%)
Few-Shot: 2/2 (100%)

Angry:
Zero-Shot: 2/2 (100%)
One-Shot: 2/2 (100%)
Few-Shot: 2/2 (100%)

Anxious:
Zero-Shot: 2/2 (100%)
One-Shot: 2/2 (100%)
Few-Shot: 2/2 (100%)

Neutral:
Zero-Shot: 2/2 (100%)
One-Shot: 2/2 (100%)
Few-Shot: 2/2 (100%)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ROWS
PS C:\Users\chanc_pyj010\OneDrive\Documents\OF_140_AB1 & C:\Users\chanc_pyj010\codgen\mathis\env\codgen-agent\python.exe "C:\Users\chanc_pyj010\OneDrive\Documents\OF_140_AB1\emotion_detection.py"
[1]:
```

Angry:

- Zero-Shot: 2/2 (100%)
- One-Shot: 2/2 (100%)
- Few-Shot: 2/2 (100%)

Anxious:

- Zero-Shot: 2/2 (100%)
- One-Shot: 2/2 (100%)
- Few-Shot: 2/2 (100%)

Neutral:

- Zero-Shot: 2/2 (100%)
- One-Shot: 2/2 (100%)
- Few-Shot: 2/2 (100%)

Impatient:

- Zero-Shot: 2/2 (100%)
- One-Shot: 2/2 (100%)
- Few-Shot: 2/2 (100%)

Surprised:

- Zero-Shot: 2/2 (100%)
- One-Shot: 2/2 (100%)
- Few-Shot: 2/2 (100%)

Sad:

- Zero-Shot: 2/2 (100%)
- One-Shot: 2/2 (100%)
- Few-Shot: 2/2 (100%)

---

**Ambiguity Handling Across Techniques:**

Zero-Shot (100% accuracy):

- ✗ Struggles with ambiguous emotions (mixed feelings)
- ✗ May struggle with nuanced emotion detection
- ✗ Hard for zero-shot to distinguish between emotions
- ✗ May confuse similar emotions (sad vs anxious)

One-Shot (98% accuracy):

- Handles ambiguity better
- Better context than zero-shot
- Still has some difficulty distinguishing between emotions
- Partial agreement in ambiguity handling

Few-Shot (100% accuracy):

- ✓ Handles ambiguity best
- ✓ Ability to learn from few shots
- ✓ Better distinction between emotions
- ✓ Aids in distinguishing between similar terms
- ✓ Most reliable for mental health support accuracy

Key Insight: Emotions often overlap (e.g., "anxious + angry", "sad + anxious")

---

**RECOMMENDATION:** Use Few-Shot Prompting for Mental Health Chatbot Section Detection

- ✗ Any emotion
- ✓ Handles ambiguous emotions
- ✓ Handles few-shot detection better
- ✓ Critical for mental health support accuracy

---

```
PS C:\Users\chanc_pyj010\OneDrive\Documents\OF_140_AB1 & C:\Users\chanc_pyj010\codgen\mathis\env\codgen-agent\python.exe "C:\Users\chanc_pyj010\OneDrive\Documents\OF_140_AB1\emotion_detection.py"
EMOTION DETECTION: ZERO-SHOT VS ONE-SHOT VS FEW-SHOT
```

---

Accuracy Summary:

Technique	Accuracy
Zero-Shot	98.00% (100%)
One-Shot	98.00% (100%)
Few-Shot	100.00% (100%)

```
[1]:
```