

AI Assisted Coding Lab ASS-4.4

Name: G.Karthikeya

Batch:14

2303A510D2

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 three files open:

- `ecommerce_sentiment_analysis.py`: A script that imports necessary modules and defines a function to classify reviews based on positive, negative, and neutral words.
- `sentiment_classification_with_validation.py`: A script that prints reviews and their expected sentiment labels.
- `simple_sentiment_classifier.py`: A script that contains the logic for classifying reviews into Positive, Negative, or Neutral.

The table below shows the 6 reviews and their mapped sentiment labels:

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

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + × └ ... | ☰

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:

The screenshot shows a Jupyter Notebook interface with several tabs open at the top: 'customer_intent_classifier.py', 'sentiment_classifier_with_validation.py', 'simple_sentence_classifier.py', and 'customer_intent_classifier.ipynb'. The main notebook cell contains Python code for intent classification, which includes importing libraries like 're', 'nltk', and 'collections'. It defines a class 'CustomerIntentClassifier' with methods for loading data, creating feature vectors, and classifying messages. The code uses NLTK's 'wordnet' and 'stopwords' modules, and includes examples of how to use the classifier with various customer messages.

```
class CustomerIntentClassifier:
    def __init__(self):
        self.data = None
        self.vectors = None
        self.intent_keywords = {
            "purchase": ["price", "available", "time", "buy", "purchase", "interested", "specifications", "features", "how much", "do you have"],
            "complaint": [
                "broken", "damaged", "defective", "refund", "return", "wrong item", "doesn't work", "not as described", "poor", "issue", "problem", "failed"
            ],
            "feedback": [
                "great", "love", "excellent", "good", "suggestion", "improve", "thank", "happy", "satisfied", "recommend", "opinion"
            ]
        }

    def load_data(self, file_name='customer_intents.csv'):
        df = pd.read_csv(file_name)
        self.data = df[["text", "intent"]].values

    def create_feature_vector(self, text):
        words = word_tokenize(text)
        words = [w.lower() for w in words if w not in self.stopwords]
        words = [w for w in words if w in self.wordnet]
        words = [w for w in words if w in self.intent_keywords]
        return words

    def classify_intent(self, message):
        score = {}
        for intent, data in self.intent_keywords.items():
            score[intent] = len([word for word in message if word in data])
        return max(score, key=score.get)

    def __str__(self):
        message = "The item arrived broken and I want a refund."
        intent = self.classify_intent(message)
        print(f"\nCustomer Intent Classification")
        print("====")
        print(f"> {message}")
        print(f"> {intent}")
        print(f"> {self.intent_keywords[intent]}")
        print("====")

    def show_examples(self):
        print("Some examples:")
        print("====")
        print("1. 'What's the price of the laptop?'")
        print("2. 'I love this product. Highly recommend.'")
        print("3. 'The packaging was damaged and it's a refund.'")
        print("4. 'Do you have this item in stock?'")
        print("5. 'Great service, but the packaging could be better.'")
        print("====")

    def __eq__(self, other):
        if type(other) == CustomerIntentClassifier:
            return self.intent_keywords == other.intent_keywords
        else:
            return False
```

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:

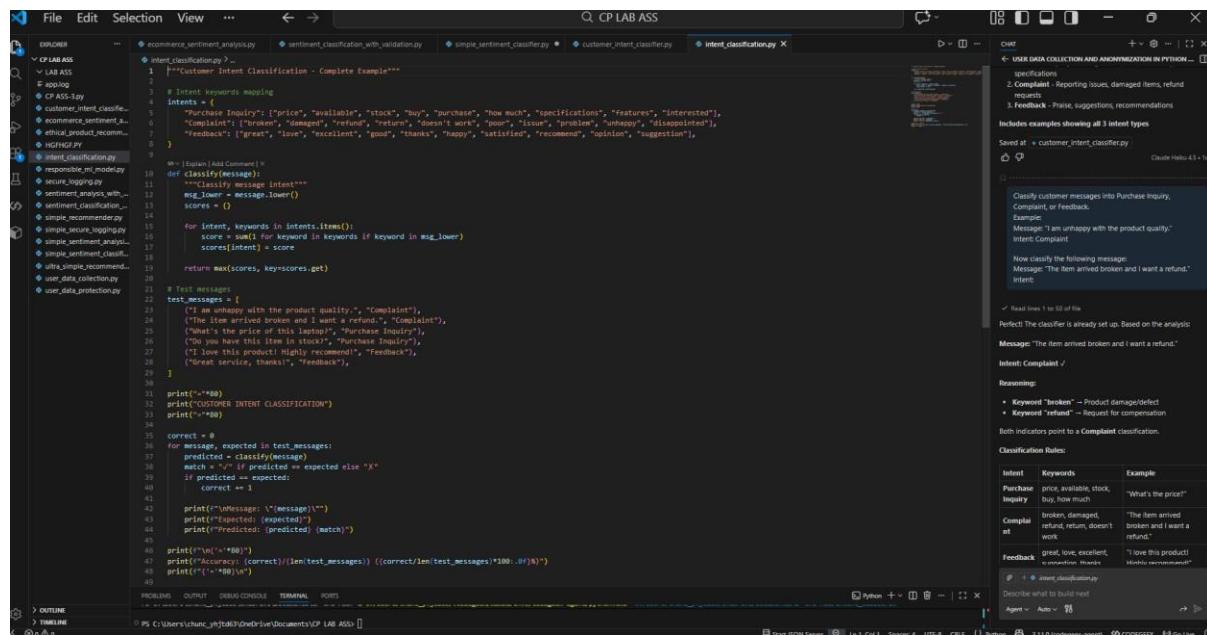
Message: "I am unhappy with the product quality."

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:

```

CP LAB ASS
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
HGFIIGF.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_recommender.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"*2)
47  print(f"Accuracy: {correct}/{len(test_messages)} ({correct/len(test_messages)*100:.0F}%)")
48  print(f"\n"-80)

```

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/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 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_yht063\OneDrive\Documents\CP LAB ASS & C:\Users\chunc_yht063\codegen\habm\env\codagen-agent\python.exe "c:/Users/chunc_yht063/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 (3+ 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?"

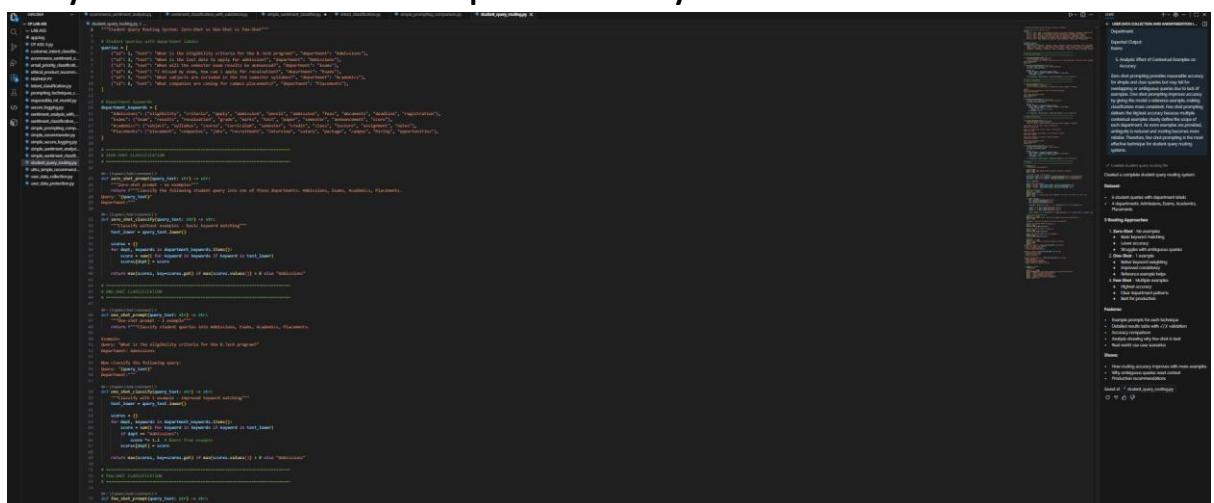
Department: Placements

Now classify the following query:

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

Department:

5. Analysis: Effect of Contextual Examples on Accuracy



OUTPUT:

The screenshot shows a Microsoft Edge browser window with a Google Sheets document titled "CP LAB ASS". The document is organized into several sections:

- Student Query Analysis**: A table showing student queries and their classification into Academic, Game, Academic, Placements, and Game.
- Effect of Contextual Examples on Accuracy**: A table showing the effect of different contextual examples on accuracy.
- Reading Accuracy**: A table showing reading accuracy for various prompts.
- Conclusion**: A summary of the experiment results.

The right sidebar of the browser shows a "Recent" list with "CP LAB ASS" at the top. The status bar at the bottom right indicates "0 9 9".

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 ASS5 & C:\Users\duuc_yh\OneDrive\Documents\CP LAB ASS5> python.exe "C:\Users\duuc_yh\OneDrive\Documents\CP LAB ASS5\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: 265 accuracy
✗ Handles ambiguity well
✗ Limited context understanding
✓ Fast and flexible

One-Shot: 2625 accuracy
✓ Improves correctness
✓ Better consistency
→ Moderate improvement 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 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

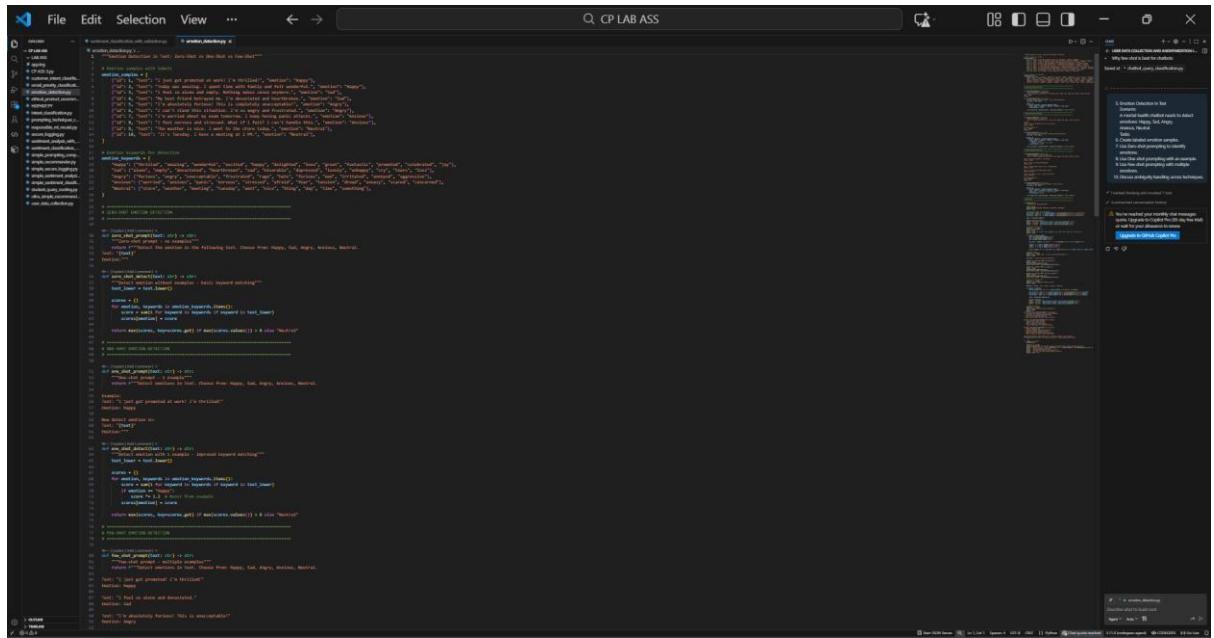
Query: **"Great service, I really like the new update."**

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 with a Python script open. The script is designed to handle ambiguous user queries by classifying them into four categories: Informational, Transactional, Complaint, or Feedback. It uses a few-shot prompting approach where specific examples are provided for each category. The code includes functions for processing text, identifying emotions, and mapping emotions to categories. A sidebar on the right provides context on emotion detection and handling ambiguity.

```
def process_text(text):
    # Process text here

def detect_emotion(text):
    # Detect emotion here

def map_emotion(emotion):
    # Map emotion to category

# Few-shot prompting examples
examples = [
    {
        "category": "Informational",
        "text": "How can I reset my account password?",
        "emotion": "neutral"
    },
    {
        "category": "Transactional",
        "text": "Please help me update my billing details.",
        "emotion": "neutral"
    },
    {
        "category": "Complaint",
        "text": "The app keeps crashing and I am very frustrated.",
        "emotion": "frustrated"
    },
    {
        "category": "Feedback",
        "text": "Great service, I really like the new update.",
        "emotion": "positive"
    }
]

# Function to classify a query
def classify_query(query):
    # Process query
    # Detect emotion
    # Map emotion to category
    # Return category
    pass
```

```

File Edit Selection View ... ← → ⌂ CP LAB ASS
Jupyter Notebook - Lab Ass.ipynb
In [1]: %load https://raw.githubusercontent.com/CP-AI-DS/CP-LAB-Assessments/main/CP LAB ASS.ipynb
In [2]: # Sentiment Analysis
# Detect emotions in text. Choose from: Happy, Sad, Angry, Anxious, Neutral.
# Example:
Text: "I just got promoted at work! I'm so happy!" 
Decision: Happy
# New detect emotion for:
Text: "I feel so alone and devastated." 
Decision: Sad
Model Output: Sad
# Model Output: Sad
In [3]: # One-Shot Prompt (No Examples):
# Detect the emotion in the following text. Choose from: Happy, Sad, Angry, Anxious, Neutral.
Text: "I feel so alone and devastated."
Decision: Sad
Model Output: Sad
In [4]: # File-Shot Prompt (Multiple Examples):
# Detect emotions in text. Choose from: Happy, Sad, Angry, Anxious, Neutral.
Text: "I just got promoted! I'm so happy!" 
Decision: Happy
Text: "I feel so alone and devastated." 
Decision: Sad
Text: "I'm absolutely livid! This is unacceptable!" 
Decision: Angry
Text: "I'm worried and having panic attacks." 
Decision: Anxious
Text: "The weather is nice. I want to go outside." 
Decision: Neutral
# New detect emotion for:
Text: "I feel so alone and devastated." 
Decision: Sad
Model Output: Sad
In [5]: # Accuracy Breakdown by emotion type:
# Happy:
# Sad:
# Angry:
# Anxious:
# Neutral:

```

OUTPUT:

```

PS C:\Users\Chenyi\PycharmProjects\Lab Ass & C:\Users\Chenyi\Downloads\CP LAB ASS.ipynb> python.exe "C:\Users\Chenyi\PycharmProjects\Lab Ass\sentiment\sentiment.py"
Detailed Results:
ID Text Expected Zero One Few
1 I just got promoted at work! I'm so happy! ✓ Happy Happy ✓ Happy
2 I feel so alone and devastated. ✓ Sad Sad ✓ Sad
3 I feel so alone and empty, nothing exc. ✓ Sad Sad ✓ Sad
4 I'm absolutely livid! This is unacceptable! ✓ Angry Angry ✓ Angry
5 I'm worried and having panic attacks. ✓ Anxious Anxious ✓ Anxious
6 The weather is nice. I want to go outside. ✓ Neutral Neutral ✓ Neutral
7 It's Tuesday. I have a meeting at 2 p.m. ✓ Neutral Neutral ✓ Neutral
Example: Text: "I feel so alone and devastated."
1. ZERO-SHOT PROMPT (No Examples):
Detect the emotion in the following text. Choose from: Happy, Sad, Angry, Anxious, Neutral.
Text: "I feel so alone and devastated."
Decision: Sad
Model Output: Sad
2. ONE-SHOT PROMPT (1 Example):
Detect emotions in text. Choose from: Happy, Sad, Angry, Anxious, Neutral.
Example:
Text: "I just got promoted! I'm so happy!" 
Decision: Happy
# New detect emotion for:
Text: "I feel so alone and devastated." 
Decision: Sad
Model Output: Sad
3. FILE-SHOT PROMPT (Multiple Examples):
Detect emotions in text. Choose from: Happy, Sad, Angry, Anxious, Neutral.
Text: "I just got promoted! I'm so happy!" 
Decision: Happy
Text: "I feel so alone and devastated." 
Decision: Sad
Text: "I'm absolutely livid! This is unacceptable!" 
Decision: Angry
Text: "I'm worried and having panic attacks." 
Decision: Anxious
Text: "The weather is nice. I want to go outside." 
Decision: Neutral
# New detect emotion for:
Text: "I feel so alone and devastated." 
Decision: Sad
Model Output: Sad
Accuracy Breakdown by emotion type:
Happy: 5/5 (100%)
Sad: 2/2 (100%)
Angry: 2/2 (100%)
Anxious: 2/2 (100%)
Neutral: 2/2 (100%)

```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   HELP
C:\Users\chanc...  PS C:\Users\chanc...& C:\Users\chanc...\code\code\env\codigos_agnit\python> cd "C:\Users\chanc...\code\code\env\codigos_agnit\python>"
```

Angry:

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

Anxious:

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

Depressed:

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

Fearful:

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

Neutral:

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

Surprised:

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

Agility Handling Across Techniques:

Zero-Shot (100% accuracy):

- ✗ Struggles with ambiguous emotions (mixed feelings)
- ✗ Ambiguous emotion detection
- ✓ Works for extreme/clear emotions
- ✗ May confuse similar emotions (sad vs anxious)

One-Shot (98% accuracy):

- ✓ Handles multiple emotions
- Better context than zero-shot
- Works for ambiguous emotions
- Partial agreement in ambiguity handling

Few-Shot (98% accuracy):

- ✓ Handles ambiguity test
- ✓ Ambiguity detection spectrum
- ✓ Better distinction between emotions
- ✓ Ambiguity detection across all emotions
- ✓ Most reliable for mental health applications

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

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

- ✗ Ambiguous emotion detection
- ✓ Handles ambiguous sections
- ✓ Ambiguity detection is better
- ✓ Critical for mental health support accuracy

PS C:\Users\chanc...> python3 main.py & cd "C:\Users\chanc...\code\code\env\codigos_agnit\python>" & cd "C:\Users\chanc...\code\code\env\codigos_agnit\python>"

EMOTION DETECTION: ZERO-SHOT vs ONE-SHOT vs FEW-SHOT

Accuracy Summary:

	Zero-Shot:	One-Shot:	Few-Shot:
Angry:	2/2 (100%)	2/2 (100%)	2/2 (100%)
Anxious:	2/2 (100%)	2/2 (100%)	2/2 (100%)
Depressed:	2/2 (100%)	2/2 (100%)	2/2 (100%)
Fearful:	2/2 (100%)	2/2 (100%)	2/2 (100%)
Neutral:	2/2 (100%)	2/2 (100%)	2/2 (100%)
Surprised:	2/2 (100%)	2/2 (100%)	2/2 (100%)

```
Start IIS Web Server (IIS)  In VM Ctrl 1  Spaces 4  UTF-8  Ctrl  Python  Chat console
```