

ASSIGNMENT-4.4

Name: N SOUMYA

Batch:13

H NO:2303A51869

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 environment with several files listed in the left sidebar, including `ecommerce_sentiment_analysis.py`, `sentiment_classification_with_validation.py`, and `simple_sentiment_classifier.py`. The main code editor contains a Python script for sentiment analysis. The code defines a `CustomerReview` class with attributes `text` and `expected`. It includes lists of positive, negative, and neutral words, and a method to calculate sentiment based on the presence of these words. A table on the right shows 6 customer reviews with their expected sentiment labels. The notebook also displays a summary of the dataset and classifier features.

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

Summary:
1. Dataset - All 6 customer reviews with expected sentiments:
o 2 Positive reviews
o 2 Neutral reviews
o 2 Negative reviews
2. Sentiment Classifier - Keyword-based analysis with:
o Positive/Negative/Neutral keyword dictionaries

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] = 0
            for keyword in data:
                if keyword in message.lower():
                    score[intent] += 1
        return max(score, key=score.get)

    def test(self):
        message = "The item arrived broken and I want a refund."
        print("Text with the provided message")
        print(message)
        print("Customer Intent Classification")
        print("-----")
        print("Message: " + message)
        print("Intent: " + self.classify_intent(message))
        print("-----")
        print("Show more examples")
        print("-----")
        print("Example 1: 'Is the price of the laptop?'")
        print("Example 2: 'I love this product. Highly recommend.'")
        print("Example 3: 'The packaging was damaged and it's a refund.'")
        print("Example 4: 'Do you have this item in stock?'")
        print("Example 5: 'Great service, but the packaging could be better.'")
        print("-----")
        for msg in examples:
            predicted_intent = self.classify_intent(msg)
            print("Message: " + msg)
            print("Intent: " + predicted_intent)
            print("-----")
```

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 Python code editor and a Jupyter Notebook interface. The code editor contains a file named `intent_classification.py` with the following content:

```

# Intent keywords mapping
intents = {
    "Purchase Inquiry": ["price", "available", "stock", "buy", "purchase", "how much", "specifications", "features", "interested"],
    "Complaint": ["broken", "damaged", "refund", "return", "doesn't work", "poor", "issue", "problem", "unhappy", "disappointed"],
    "Feedback": ["great", "love", "excellent", "good", "thanks", "happy", "satisfied", "recommend", "opinion", "suggestion"]
}

# Test messages
test_messages = [
    ("I am unhappy with the product quality.", "Complaint"),
    ("The item arrived broken and I want a refund.", "Complaint"),
    ("What's the price of this laptop?", "Purchase Inquiry"),
    ("Do you have this item in stock?", "Purchase Inquiry"),
    ("I love this product! Highly recommend!", "Feedback"),
    ("Great service, thanks!", "Feedback"),
    ("Great service, thanks!", "Feedback"),
    ("Great service, thanks!", "Feedback")
]

print("CUSTOMER INTENT CLASSIFICATION")
for message in test_messages:
    predicted = classify(message)
    print(f"\nMessage: {message}\nPredicted: {predicted}\n")
    if predicted == expected:
        correct += 1

print(f"\nAccuracy: {(correct/len(test_messages)) * 100:.2f}%")

```

The Jupyter Notebook cell contains the following text:

```

customer_intent_classifier.py
-----
Classify customer messages into Purchase Inquiry, Complaint, or Feedback.

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

How classify the following message:
Message: "The item arrived broken and I want a refund."
Intent: Complaint

✓ Read lines 1 to 10 of file
Perfect! The classifier is already set up. Based on the analysis:
Message: "The item arrived broken and I want a refund."
Intent: Complaint ✓

Reasoning:
• Keyword "broken" - Product damage/defect
• Keyword "refund" - Request for compensation
Both indicators point to a Complaint classification.

Classification Rules:

```

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, good, thanks, happy, satisfied, recommend, opinion, suggestion	"I love this product! Highly recommend!"

Below the table, there is a note: "Describe what to build next".

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
HGHIIGF.PY
intent_classification.py
responsible_ml_model.py
secure_logging.py
sentiment_analysis_with_ml.py
sentiment_classification.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", "disappointed"],
7      "Feedback": ["great", "love", "excellent", "good", "thanks", "happy", "satisfied", "recommend", "opinion", "suggestion"],
8  }
9
10  # [Explain] Add Comment]
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("*"*80)
33  print("CUSTOMER INTENT CLASSIFICATION")
34  print("*"*80)
35
36  correct = 0
37  for message, expected in test_messages:
38      predicted = classify(message)
39      match = "✓" if predicted == expected else "X"
40      if predicted == expected:
41          correct += 1
42
43  print("\nMessage: \"{}\"".format(message))
44  print("Expected: {}".format(expected))
45  print("Predicted: {} ({}{})".format(predicted, match))
46
47  print("\n" * 80)
48  print("Accuracy: {} ({})".format(correct, len(test_messages)))
49  print(" "*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.

The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** File, Edit, Selection, View, ..., back, forward, search bar (CP LAB ASS).
- Left Sidebar:** Shows a file tree with several Java files under "src/main/java".
- Code Editor:** Displays Java code for a class named "Complexity". The code includes imports for `java.util.List`, `java.util.ArrayList`, and `java.util.LinkedList`. It contains methods for calculating complexity based on input parameters like `methodCount`, `loopDepth`, and `recursionDepth`. It also includes a `main` method with a `for` loop testing various input values.
- Right Sidebar:** Contains developer tools:
 - Elements:** Shows the DOM structure of the page.
 - Console:** Shows log messages related to Java code execution.
 - Performance:** Shows CPU usage over time.
 - Network:** Shows network requests.
 - Resources:** Shows file resources.
 - Elements:** Shows element details.
 - Elements:** Shows element styles.
 - Elements:** Shows element shadows.
 - Elements:** Shows element tree.
 - Elements:** Shows element metrics.
 - Elements:** Shows element metrics.

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

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\chuc_yhjt63\OneDrive\Documents\CP LAB ASS & C:\Users\chuc_yhjt63\.codegen\meta\envs\codegen-agent\python.exe "c:/Users/chuc_yhjt63/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 (>1 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 displays 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 categories.
- Effect of Contextual Examples on Accuracy**: A table showing the effect of different contextual examples on reading accuracy.
- Reading Accuracy**: A table showing reading accuracy for various prompts.
- Annotations**: A section with annotations such as "Effect of Contextual Examples on Accuracy" and "Effect of Department on Accuracy".
- Information**: A section with notes about department names and reading accuracy.
- RECOMMENDATION**: A section with recommendations for improving student query ranking.

The right side of the screen shows a "Recent" list of files and a "Copilot" interface with a "Copilot" button and a "Copilot" tab.

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

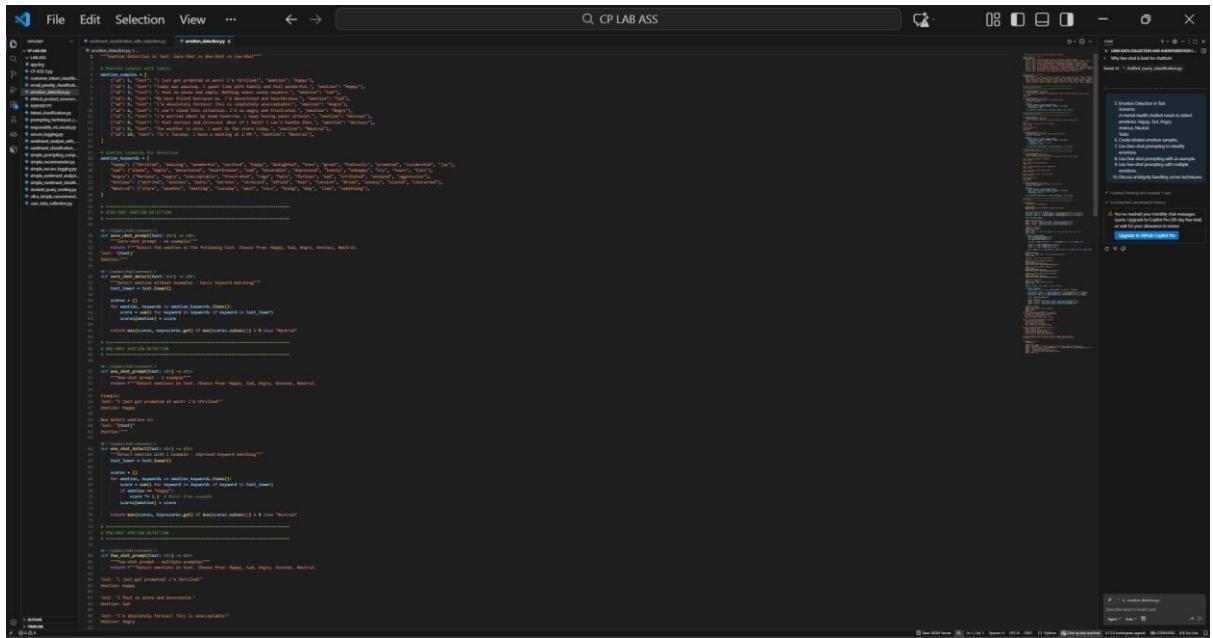
Query: ***"Great service, I really like the new update."***

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 Python code. The code defines several functions for processing text and classifying emotions. It uses regular expressions to find words related to specific emotions in the input text. The `get_emotions` function takes a text string and returns a dictionary of emotion counts. The `get_emotion` function takes a text string and returns the most frequent emotion. The `get_emotion_label` function takes a text string and returns a label based on the emotion count. The `get_emotion_labels` function takes a text string and returns a list of emotion labels. The `get_emotion_label_for_text` function takes a text string and returns a single emotion label. The `get_emotion_label_for_text` function is annotated with a docstring explaining it handles multiple emotions by selecting the most frequent one. The code also includes a `main` function that prints the emotion label for a sample text.

```
def get_emotions(text):
    """Get emotions from text. Returns a dict of emotions and their counts.
    Args:
        text (str): The text to analyze.
    Returns:
        dict: A dictionary where keys are emotions and values are counts.
    """
    emotion_counts = {}
    for emotion, words in EMOTIONS.items():
        if any(word in text for word in words):
            emotion_counts[emotion] = emotion_counts.get(emotion, 0) + 1
    return emotion_counts

def get_emotion(text):
    """Get the most frequent emotion from text.
    Args:
        text (str): The text to analyze.
    Returns:
        str: The most frequent emotion.
    """
    emotion_counts = get_emotions(text)
    max_emotion = max(emotion_counts, key=emotion_counts.get)
    return max_emotion

def get_emotion_label(text):
    """Get the emotion label from text.
    Args:
        text (str): The text to analyze.
    Returns:
        str: The emotion label.
    """
    emotion_counts = get_emotions(text)
    max_emotion = max(emotion_counts, key=emotion_counts.get)
    return max_emotion

def get_emotion_labels(text):
    """Get all emotion labels from text.
    Args:
        text (str): The text to analyze.
    Returns:
        list: A list of emotion labels.
    """
    emotion_counts = get_emotions(text)
    emotion_labels = [emotion for emotion, count in emotion_counts.items() if count > 0]
    return emotion_labels

def get_emotion_label_for_text(text):
    """Get the emotion label for text. Handles multiple emotions by selecting the most frequent one.
    Args:
        text (str): The text to analyze.
    Returns:
        str: The emotion label.
    """
    emotion_labels = get_emotion_labels(text)
    if len(emotion_labels) == 1:
        return emotion_labels[0]
    else:
        return "mixed"

def main():
    text = "I'm absolutely热爱 this environment! I'm so happy."
    print(get_emotion_label_for_text(text))

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

OUTPUT:

The screenshot shows a Jupyter Notebook interface with several code cells and output sections. The code cells contain Python code for sentiment analysis, and the output sections show the results for different text snippets. The results are presented in tables with columns for 'Text', 'Expected', 'Zero', 'One', and 'Few'. The 'One' column is highlighted in yellow, indicating the predicted sentiment.

Code Cells:

```
from vaderSentiment import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()
# Load dataset
df = pd.read_csv('sentiment.csv')
# Check first few rows
df.head()
# Define function to get sentiment
def get_sentiment(text):
    # Create a sentiment intensity analyzer
    sid = SentimentIntensityAnalyzer()
    # Compute scores
    scores = sid.polarity_scores(text)
    # Get overall polarity
    if scores['compound'] > 0.05:
        return 'Positive'
    elif scores['compound'] < -0.05:
        return 'Negative'
    else:
        return 'Neutral'
```

Output Section 1: Detailed Results

ID	Text	Expected	Zero	One	Few
1	I just got promoted at work! I'm ...	Happy	✓ Happy	✓ Happy	✓ Happy
2	Today was amazing. I spent time with ...	Happy	✓ Happy	✓ Happy	✓ Happy
3	I'm so happy to be here.	Happy	✓ Happy	✓ Happy	✓ Happy
4	My best friend introduced me. I'm ...	Sad	✓ Sad	✓ Sad	✓ Sad
5	I'm ...	Angry	✓ Angry	✓ Angry	✓ Angry
6	I can't stand this situation. I'm ...	Angry	✓ Angry	✓ Angry	✓ Angry
7	I'm ...	Anxious	✓ Anxious	✓ Anxious	✓ Anxious
8	I just started my new job.	Anxious	✓ Anxious	✓ Anxious	✓ Anxious
9	I just started my new job.	Anxious	✓ Anxious	✓ Anxious	✓ Anxious
10	The weather is nice. I want to go to the ...	Neutral	✓ Neutral	✓ Neutral	✓ Neutral
11	The weather is nice. I have a meeting at the ...	Neutral	✓ Neutral	✓ Neutral	✓ Neutral

Output Section 2: Example 1

1. ONE-SHOT PROMPT (1 Example)

Text: "I feel as alone and devastated."

Text: "I feel as alone and devastated."
Section: None
Model Output: Sad

Output Section 3: Example 2

1. ONE-SHOT PROMPT (1 Example)

Text: sentences in text. Choose from: Happy, Sad, Angry, Anxious, Neutral.

Text: "I just got promoted at work! I'm ..."

Text: "I just got promoted! I'm ..."

Text: "I feel as alone and devastated."

Text: "I feel as alone and devastated."

Text: "I'm absolutely furious! This is unacceptable!"

Text: "I'm worried and having panic attacks."

Text: "The weather is nice. I want to go to the store."

Text: "I feel as alone and devastated."

Text: "I feel as alone and devastated."

Text: "I'm ..."

Output Section 4: Accuracy Breakdown by Section Type

Section Type	Happy	Sad	Angry	Anxious	Neutral
Zero-Shot	2/2 (100%)	0/2 (0%)	0/2 (0%)	0/2 (0%)	2/2 (100%)
One-Shot	2/2 (100%)	0/2 (0%)	0/2 (0%)	0/2 (0%)	2/2 (100%)
Two-Shot	2/2 (100%)	0/2 (0%)	0/2 (0%)	0/2 (0%)	2/2 (100%)
Three-Shot	2/2 (100%)	0/2 (0%)	0/2 (0%)	0/2 (0%)	2/2 (100%)

```
PROBLEMS OUTPUT STATUS CHECK TERMINAL HELP
File C:\Users\charles_pyj\Downloads\Documents\CF_LM_RSS & C:\Users\charles_pyj\Downloads\codegen\mental_health\agent\python\src \t\Answers\charles_pyj\Downloads\Documents\CF_LM_RSS\sentiment_detection.py
C:\Users\charles_pyj\Downloads\Documents\CF_LM_RSS > C:\Users\charles_pyj\Downloads\codegen\mental_health\agent\python\src \t\Answers\charles_pyj\Downloads\Documents\CF_LM_RSS\sentiment_detection.py

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%)

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

HAPPY:
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%)

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

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

Ambiguity Handling Across Techniques:
Zero-Shot (100% accuracy):
X Struggles with ambiguous sections (Mixed Feelings)
X Ambiguous sections can be detected
X Works for similar sections
X May confuse similar sections (sad vs anxious)

One-Shot (98% accuracy):
- Handles ambiguous sections
- Better suited than zero-shot
- Works for similar sections
- Partial agreement in ambiguity handling

Few-Shot (100% accuracy):
✓ Handles ambiguity test
✓ Ambiguous sections are better
✓ Better distinction between sections
✓ Ambiguous sections are better
✓ Most reliable for mental health support accuracy

Any Insight: Sections often overlap (e.g., "anxious + angry", "sad + anxious")
Few-shot processing provides the clearest patterns for distinguishing these nuances.

RECOMMENDATION: Use Few-Shot Processing for Mental Health Chatbot Section Detection
X Anxious
X Angry
X Sad
X Neutral
X Surprise
X Mixed
X Confused
X Critical
X Critical for mental health support accuracy

PROBLEM DETECTION: Zero-Shot vs One-Shot vs Few-Shot
Accuracy Summary:
Zero-Shot: 2/2 (100%)
One-Shot: 2/2 (100%)
Few-Shot: 2/2 (100%)
```