

AI Assisted Coding Lab ASS-4.4

Name: D. Nithin

Batch: 13
2303A51845

1. Sentiment Classification for Customer Reviews

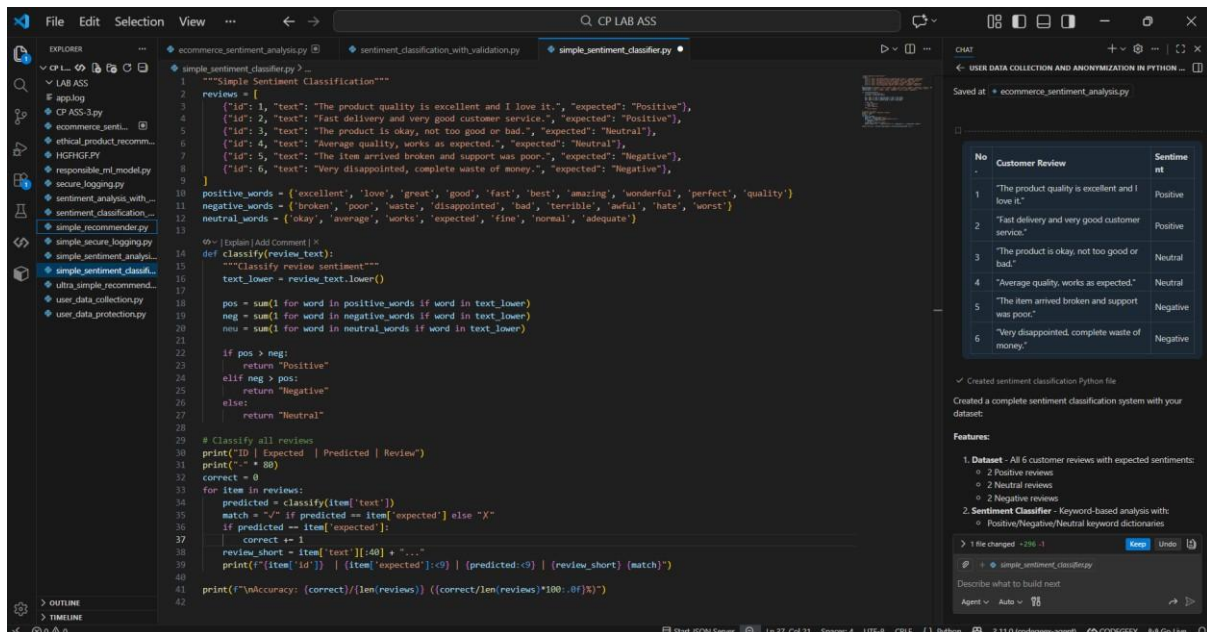
Scenario:

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



```
1 """Simple Sentiment Classification"""
2 reviews = [
3     {"id": 1, "text": "The product quality is excellent and I love it.", "expected": "Positive"},
4     {"id": 2, "text": "Fast delivery and very good customer service.", "expected": "Positive"},
5     {"id": 3, "text": "The product is okay, not too good or bad.", "expected": "Neutral"},
6     {"id": 4, "text": "Average quality, works as expected.", "expected": "Neutral"},
7     {"id": 5, "text": "The item arrived broken and support was poor.", "expected": "Negative"},
8     {"id": 6, "text": "Very disappointed, complete waste of money.", "expected": "Negative"},
9 ]
10 positive_words = {'excellent', 'love', 'great', 'good', 'fast', 'best', 'amazing', 'wonderful', 'perfect', 'quality'}
11 negative_words = {'broken', 'poor', 'waste', 'disappointed', 'bad', 'terrible', 'awful', 'hate', 'worst'}
12 neutral_words = {'okay', 'average', 'works', 'expected', 'fine', 'normal', 'adequate'}
13
14 def classify(review_text):
15     """Classify review sentiment"""
16     text_lower = review_text.lower()
17
18     pos = sum(1 for word in positive_words if word in text_lower)
19     neg = sum(1 for word in negative_words if word in text_lower)
20     neu = sum(1 for word in neutral_words if word in text_lower)
21
22     if pos > neg:
23         return "Positive"
24     elif neg > pos:
25         return "Negative"
26     else:
27         return "Neutral"
28
29 # Classify all reviews
30 print("ID | Expected | Predicted | Review")
31 print("-" * 80)
32 correct = 0
33 for item in reviews:
34     predicted = classify(item['text'])
35     match = "✓" if predicted == item['expected'] else "X"
36     if predicted != item['expected']:
37         correct += 1
38     review_short = item['text'][:40] + "..."
39     print(f"{item['id']} | {item['expected']} | {predicted} | {review_short} {match}")
40
41 print(f"\nAccuracy: {correct}/{len(reviews)} ({correct/len(reviews)*100:.0f}%)")
```

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:


```
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:\Users\chunc_yhjt63\.codegeex\mamba\envs\codegeex-agent\python.exe "c:/Users/chunc_yhjt63/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_yhjt63\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 VS Code editor with a Python script named `intent_classification.py` in the `CP LAB ASS` project. The script defines a `classify_message` function that uses a keyword-based approach to classify customer messages into three categories: Purchase Inquiry, Complaint, or Feedback. It includes a list of keywords for each intent and a scoring system to determine the most likely intent. The script also includes test messages and a main execution block that prints the results of the classification.

On the right side of the editor, there is a chat window titled `CHAT` showing the results of the classification for the message `"The item arrived broken and I want a refund."`. The chat window displays the following information:

- Specifications:** 1. Complaint - Reporting issues, damaged items, refund requests; 2. Complaint - Reporting issues, damaged items, refund requests; 3. Feedback - Praise, suggestions, recommendations.
- Includes examples showing all 3 intent types**
- Saved at:** `customer_intent_classifier.py`
- 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: 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, return, doesn't work	"The item arrived broken and I want a refund."
Feedback	good, love, excellent, awesome, thanks	"I love this product! Highly recommend!"

OUTPUT:

```

PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjt63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjt63/OneDrive\Documents\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_yhjt63\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:

```

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 def classify(message):
11     """Classify message intent"""
12     msg_lower = message.lower()
13     scores = {}
14
15     for intent, keywords in intents.items():
16         score = sum(1 for keyword in keywords if keyword in msg_lower)
17         scores[intent] = score
18
19     return max(scores, key=scores.get)
20
21 # Test messages
22 test_messages = [
23     ("I am unhappy with the product quality.", "Complaint"),
24     ("The item arrived broken and I want a refund.", "Complaint"),
25     ("What's the price of this laptop?", "Purchase Inquiry"),
26     ("Do you have this item in stock?", "Purchase Inquiry"),
27     ("I love this product! Highly recommend!", "Feedback"),
28     ("Great service, thanks!", "Feedback"),
29 ]
30
31 print("\n=====")
32 print("CUSTOMER INTENT CLASSIFICATION")
33 print("=====")
34
35 correct = 0
36 for message, expected in test_messages:
37     predicted = classify(message)
38     match = "✓" 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\n")
47 print(f"Accuracy: {correct}/{len(test_messages)} ({correct/len(test_messages)*100:.0f}%)")
48 print("\n")
49

```

OUTPUT:

```

PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> ^C
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjt63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjt63/OneDrive\Documents\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_yhjt63\OneDrive\Documents\CP LAB ASS>

```

E) Compare the outputs and discuss accuracy differences.

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

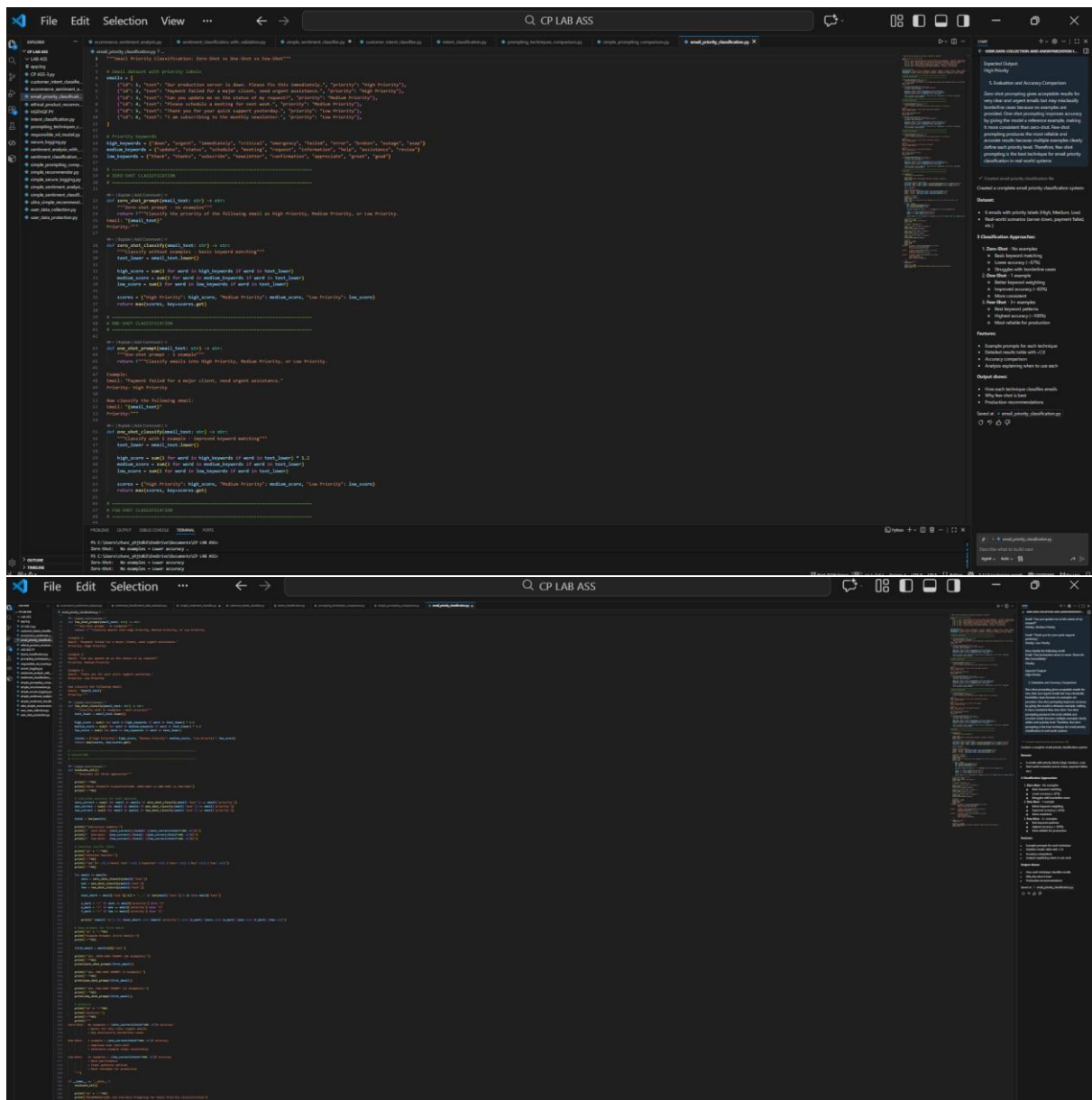
Now classify the following email:

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\churc_jh163\OneDrive\Documents\CP LAB A55> & C:\Users\churc_jh163\OneDrive\Documents\CP LAB A55\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:

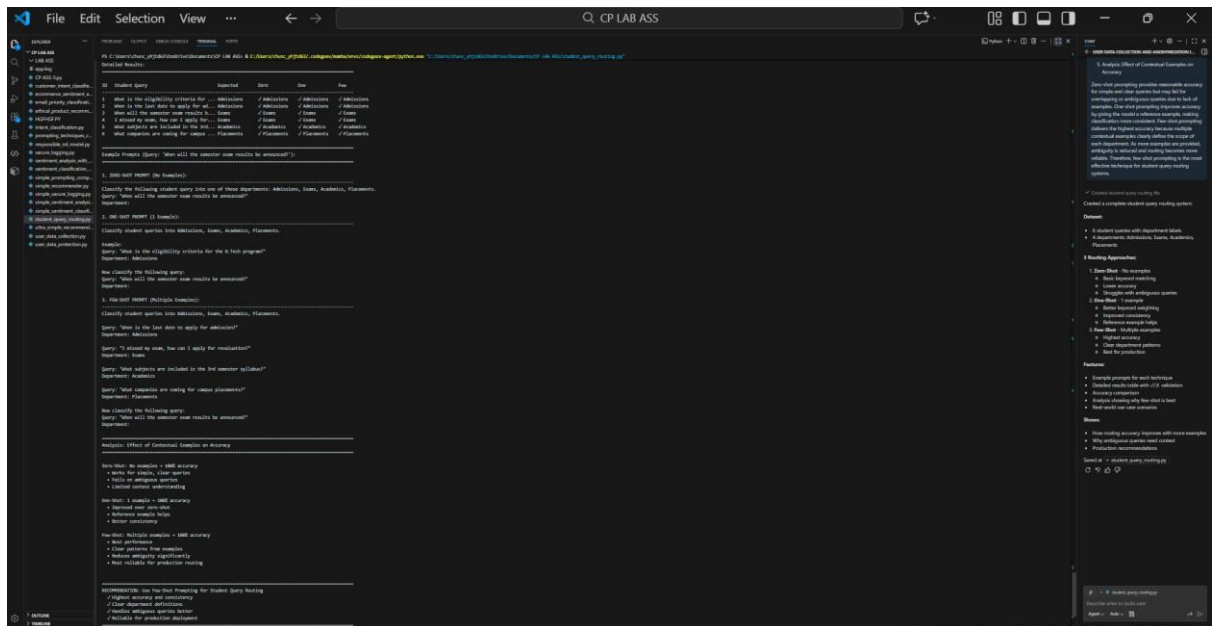
5. Analysis: Effect of Contextual Examples on Accuracy

The image displays two screenshots of a Jupyter Notebook, illustrating the effect of adding contextual examples on the accuracy of a query classification model. The notebook is written in Python and uses the `sklearn` library for machine learning.

Top Screenshot: This screenshot shows the initial code. It defines a function `classify_query` that takes a query string as input and returns a classification result. The function uses a `DecisionTreeClassifier` to classify the query. The code includes a list of queries and their corresponding department labels. The output of the function is displayed as a string.

Bottom Screenshot: This screenshot shows the code after adding contextual examples. The function `classify_query` is updated to include a list of contextual examples (queries and their corresponding department labels) that are used to train the classifier. The output of the function is displayed as a string, showing the result of the classification.

OUTPUT:



4. Chatbot Question Type Detection Scenario:

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

1. Prepare 6 chatbot queries mapped to question types.

2. Design prompts for Zero-shot, One-shot, and Few-shot learning.

Zero-Shot Prompt

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

Query: "I want to cancel my subscription."

One-Shot Prompt

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

Example:

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

Question Type: Informational Now

classify the following query:

Query: "I want to cancel my subscription."

Few-Shot Prompt

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

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

Question Type: Informational

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

Question Type: Transactional

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

Question Type: Complaint

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

Question Type: Feedback

Now classify the following query:

Query: "I want to cancel my subscription."

3. Test all prompts on the same unseen queries.

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

4. Compare response correctness and ambiguity handling.

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

6. Document observations.



OUTPUT:

```

PS C:\Users\churc_ghjtd5\OneDrive\Documents\CP LAB A&S & C:\Users\churc_ghjtd5\.config\haila\ana\chatgpt-agent\python.exe "C:\Users\churc_ghjtd5\OneDrive\Documents\CP LAB A&S\chatbot_query_classification.py"

Example Prompts (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

=====
Comparison: Response Correctness and Ambiguity Handling

Zero-Shot: 30% accuracy
- Struggles with ambiguous queries
- Limited context understanding
- Fast and flexible

One-Shot: 50% accuracy
- Improves correctness
- Better consistency
- Moderate improvement over zero-shot

Few-Shot: 80% accuracy
- Best accuracy and consistency
- Handles ambiguity well
- Clear patterns from examples
- Most reliable for production

=====
Observations
=====

1. Few-shot gives most accurate results (80%)
2. One-shot offers moderate improvement over zero-shot
3. Zero-shot is fast but less reliable for complex queries
4. More examples significantly improve accuracy
5. Multiple examples reduce confusion for ambiguous queries
6. Few-shot recommended for production chatbots

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

PS C:\Users\churc_ghjtd5\OneDrive\Documents\CP LAB A&S >

```

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

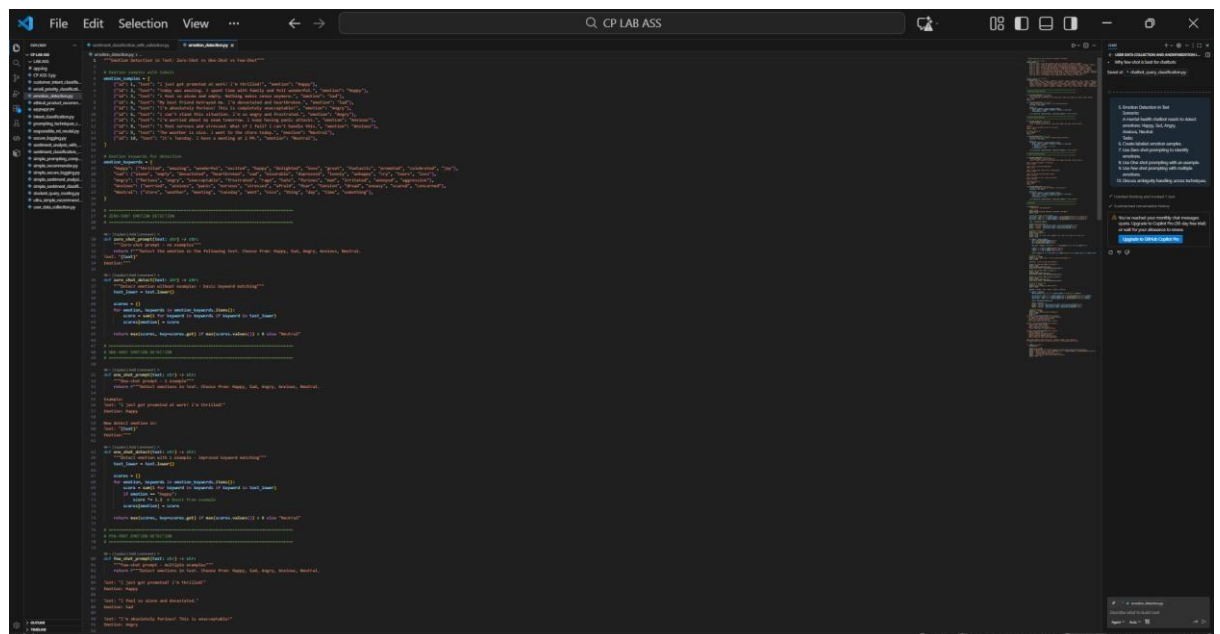
Query: *"Great service, I really like the new update."*

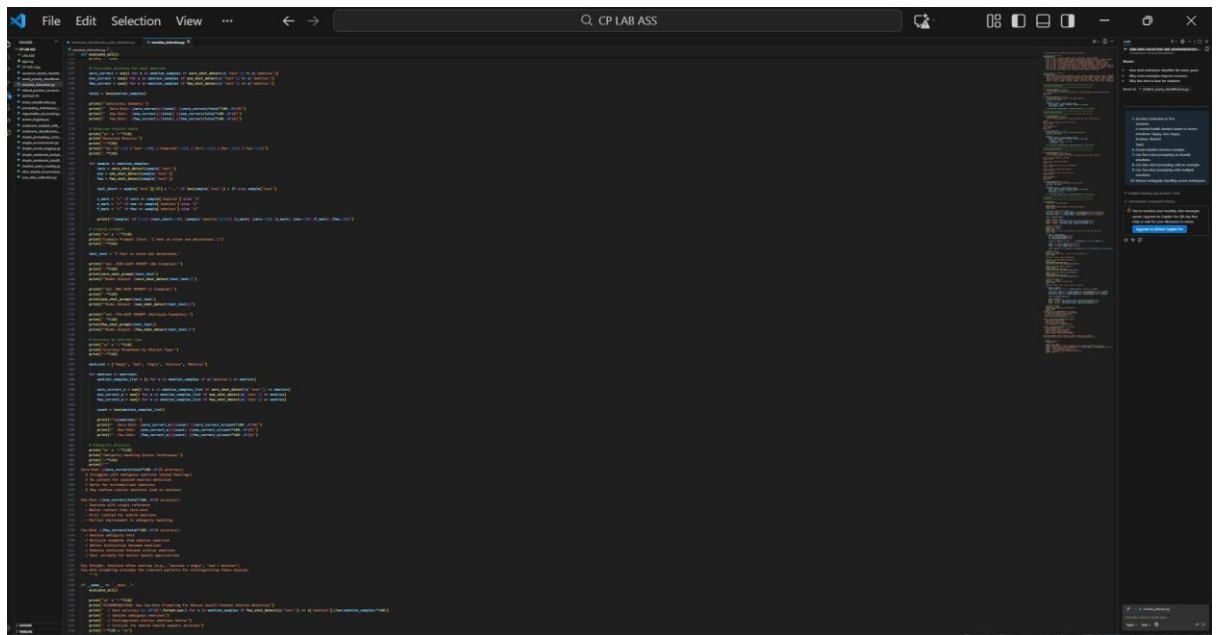
Question Type: Feedback

Now classify the following query:

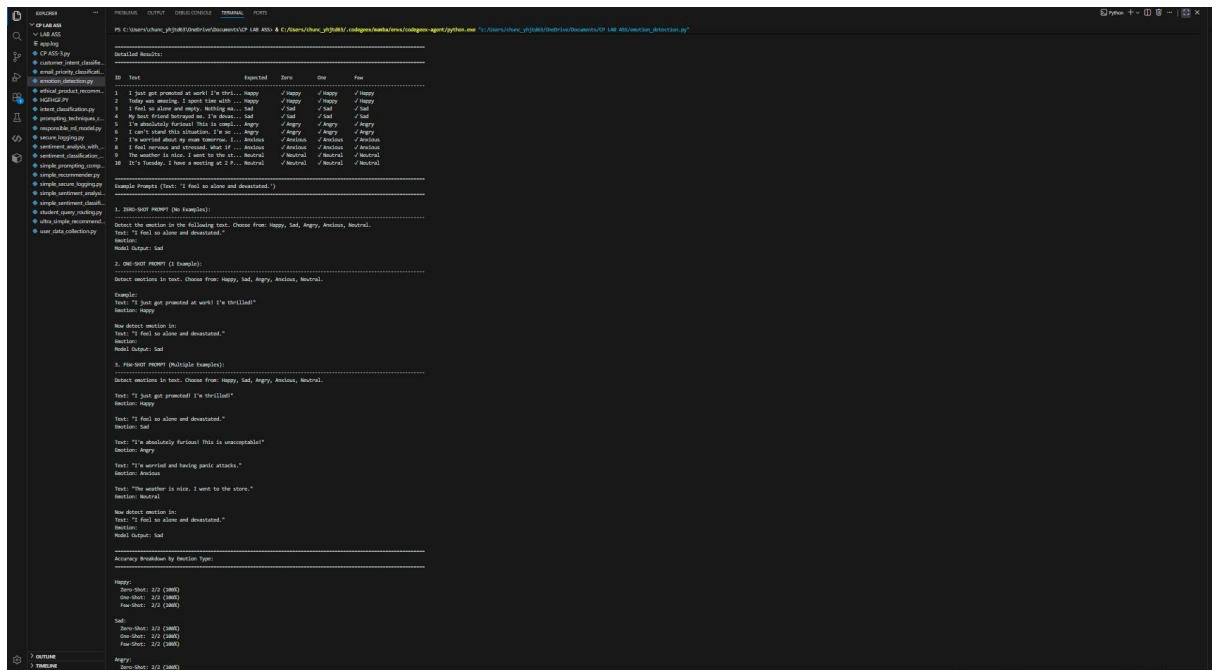
Query: *"I want to cancel my subscription."*

5. Discuss ambiguity handling across techniques.





OUTPUT:



[illegible]