

2303A51502

Batch-25

Assignment-4.4

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.

Tasks:

- a) Prepare 6 short customer reviews mapped to sentiment labels.
- b) Design a Zero-shot prompt to classify sentiment.
- c) Design a One-shot prompt with one labeled example.
- d) Design a Few-shot prompt with 3–5 labeled examples.
- e) Compare the outputs and discuss accuracy differences.

The screenshot shows the AI Assted IDE interface. On the left, the Explorer sidebar lists various files including 'ass 4.4.py' which is currently open. The code in 'ass 4.4.py' performs sentiment analysis on a list of reviews. It prints each review with its ID and sentiment label. It then counts the number of positive, negative, and neutral reviews. Finally, it prints a summary with the counts. The terminal output shows five sample reviews with their respective sentiment labels (Positive, Negative, or Neutral). A right-hand panel displays a snippet titled 'SENTIMENT CLASSIFICATION FOR CUSTOMER REVIEWS' with sample code and instructions.

```
24     'review': "It's okay. Does what it's supposed to do, nothing special."
25     'sentiment': 'Neutral'
26   }
27 }
28 ]
29 # Print reviews with sentiment labels
30 print(*= 20)
31 print("E-commerce Customer Reviews - Sentiment Analysis")
32 print(*= 70)
33 print(*= 70)
34 for idx, item in enumerate(reviews, 1):
35   print("\nReview #", idx)
36   print("Text:", item['review'])
37   print("Sentiment:", item['sentiment'])
38   print(*= 70)
39
40 print("Summary:")
41 positive_count = sum(1 for item in reviews if item['sentiment'] == 'Positive')
42 negative_count = sum(1 for item in reviews if item['sentiment'] == 'Negative')
43 neutral_count = sum(1 for item in reviews if item['sentiment'] == 'Neutral')
44
45 print(f"Positive: {positive_count} | Negative: {negative_count} | Neutral: {neutral_count}")
46
```

This screenshot shows the same AI Assted IDE interface, but with a different Python file open: 'ass 4.4.4.py'. This code implements a simple keyword-based sentiment classifier. It defines a function 'classify_sentiment' that takes a review as input and returns 'Positive', 'Negative', or 'Neutral' based on keyword counts. The code then tests this classifier with user input and existing reviews. The terminal output shows the classifier correctly identifying the sentiment of a user-reviewed item and existing reviews. A right-hand panel displays a snippet titled 'SENTIMENT CLASSIFICATION FOR CUSTOMER REVIEWS' with sample code and instructions.

```
54 def classify_sentiment(review):
55   if positive_keywords > negative_count:
56     return "Positive"
57   elif negative_count > positive_count:
58     return "Negative"
59   else:
60     return "Neutral"
61
62
63 # Test the classifier with user input
64 user_review = input("Enter a customer review: ")
65 sentiment = classify_sentiment(user_review)
66 print(*= 70)
67 print("Customer Review Sentiment Classifier")
68 print(*= 70)
69
70 # Optional: Test classifier on existing reviews
71 print(*= 70)
72 print("Testing Classifier on Existing Reviews")
73 print(*= 70)
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows files like `ai-assisted.ipynb`, `check_leap_year.py`, `leap_4.3.ipynb`, `leap_4.3.pdf`, `leap_4.3.py`, `leap_4.3_word.docx`, `leap_4.4.ipynb`, `leap_assignment-1.ipynb`, `leap_assignment-2,3.ipynb`, and `leap.year.py`.
- Code Editor:** Displays Python code for a "Customer Review Sentiment Classifier". The code includes functions for classifying reviews, printing example classifications, and a main loop for user input and classification.
- Terminal:** Shows command-line output from running the code in a Jupyter notebook environment.
- Output:** Shows the results of the sentiment classification logic.
- Timeline:** Shows a history of changes made to the file.
- Activity Bar:** Includes icons for powershell, Python, and Jupyter.
- Right Sidebar:** Titled "SENTIMENT CLASSIFICATION FOR CUSTOMER REVIEWS", it contains:
 - Example:** "The product is excellent and works perfectly" - Positive
 - "The item is okay, not great" - Neutral
 - "Very disappointed with the quality" - Negative
 - "Average experience overall" - Neutral

Text below the examples reads: "Use these examples to guide the logic. Do not use external libraries or ML models."
- Status Bar:** Shows the current file path as `C:\Users\parisa\OneDrive\Desktop\AI Assisted\ai-assisted.ipynb`, the line number as Ln 103, the column number as Col 46, the space count as Spaces 4, and the file type as UTF-8 - CR LF - Python - 3.137.

The screenshot shows the Microsoft Visual Studio Code interface. On the left is the Explorer sidebar with files like 'AI Assisted', 'add.py', 'ass 4.4.py', 'check_leap_year.py', 'lab 4.3 word.docx', 'lab 4.3 word.pdf', 'lab assignment 3.3.pdf', 'lab assignment 1.4.pdf', 'lab assignment 2.3.pdf', and 'leap_year.py'. The main editor area contains Python code for sentiment analysis, including sections for 'ZERO-SHOT APPROACH', 'ONE-SHOT APPROACH', and 'FEW-SHOT APPROACH'. The code uses enumerate to iterate through reviews, classifies them, and compares predicted vs actual sentiment. It also calculates accuracy. The right side of the interface features a ChatGPT-like AI Assistant pane titled 'SENTIMENT CLASSIFICATION FOR CUST...'. The AI has provided a summary of the code's purpose and a comparison between zero-shot, one-shot, and few-shot logic approaches. Below the AI pane, there is a code editor with a snippet of test data for sentiment analysis.

```

ss 4.4.py 9
...
309 print("Detailed Results:")
310 print("-" * 80)
311
312 for idx, item in enumerate(test_reviews, 1):
313     predicted = classify_sentiment(item['review'])
314     actual = item['sentiment']
315     if predicted == actual:
316         correct_predictions += 1
317
318     match_symbol = "/" if is_correct else "?"
319     status = "CORRECT" if is_correct else "INCORRECT"
320
321     print(f"\nText: {item['review']} | Predicted: {predicted} | Actual: {actual}")
322
323     print(f"Match Symbol: {match_symbol} | Status: {status}")
324
325     print("-" * 80)
326
327 # calculate accuracy
328 accuracy = (correct_predictions / total_reviews) * 100
329
330 print(f"\n{accuracy:.2f}%")
331 print("ACCURACY SUMMARY")
332 print("-" * 80)
333 print(f"Total Reviews Tested: {total_reviews}")
334 print(f"Correct Predictions: {correct_predictions}")
335 print(f"Incorrect Predictions: {total_reviews - correct_predictions}")
336 print(f"Accuracy Rate: {accuracy:.2f}%")
337
338 # Approach comparison
339 print("\nCLASSIFICATION APPROACH COMPARISON")
340 print("-" * 80)
341
342 comparison = """
343 ZERO-SHOT APPROACH:
344 - No examples or training provided to classifier
345 - Only pre-defined keyword lists only
346 - Fastest execution
347 - Limited to keyword presence/absence
348 - Accuracy: ~70-80% (depends on keyword completeness)
349 """
350
351 ONE-SHOT APPROACH:
352 - Used a single example to guide classification
353 - Classifier learns from one labeled instance
354 - Better context awareness than zero-shot
355 - Requires example selection and integration
356 - Accuracy: ~80-85%
357
358 FEW-SHOT APPROACH:
359 - Similar to one-shot, but uses multiple labeled instances
360 - Faster execution
361 - Better context awareness
362 - Requires example selection and integration
363 - Accuracy: ~85-90%
364
365 # Test dataset with more reviews
test_reviews = [
    {
        "review": "The product is great!",
        "sentiment": "Positive",
        "category": "Existing"
    },
    {
        "review": "Terrible experience",
        "sentiment": "Negative",
        "category": "Existing"
    },
    {
        "review": "The product arrived quickly.",
        "sentiment": "Neutral",
        "category": "Existing"
    },
    {
        "review": "Love it! Excellent quality.",
        "sentiment": "Positive",
        "category": "Existing"
    },
    {
        "review": "Not satisfied with the service.",
        "sentiment": "Negative",
        "category": "Existing"
    },
    {
        "review": "It's okay. Does the job.",
        "sentiment": "Neutral",
        "category": "Existing"
    }
]

```

2. Email Priority Classification

Scenario:

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

Tasks:

1. Create 6 sample email messages with priority labels.
2. Perform intent classification using Zero-shot prompting.
3. Perform classification using One-shot prompting.
4. Perform classification using Few-shot prompting.
5. Evaluate which technique produces the most reliable results and why.

File Edit Selection View Go Run Terminal Help

EXPLORER

- AI Assisted
- AI ass1.pdf
- AI ass1.pdf
- ass4.pdf
- check leap.yarpy
- lab 4.3 word.docx
- lab 4.3 word.pdf
- lab 4.3 yarpy
- lab 4.3 yarpy
- lab assignment 3.3.pdf
- lab assignment 1.4.pdf
- lab assignment 1-3.pdf
- leap.yarpy

ass 4.4.py

```

1 # Simple list of email tuples: (subject, body, priority)
2 emails = [
3     ("Urgent: System Outage - Immediate Action Required", "The main database server is down. All operations are halted.", "High"),
4     ("Q1 Budget Review Meeting - Next Friday at 2 PM", "Please review the attached budget documents.", "Medium"),
5     ("Office Lunch - Catering Menu For Next Week", "Please vote on your preferred lunch option.", "Low"),
6     ("Monthly Team Updates - Please Submit by End of Week", "Submit your monthly progress report by Friday.", "Medium"),
7     ("Office Supplies Restocking - New Printer Paper Available", "New printer paper has arrived in the supply closet.", "Low")
8 ]
9
10
11 # Print emails with priority
12 print("OFFICE EMAILS")
13 print("-" * 80)
14
15 for idx, (subject, body, priority) in enumerate(emails, 1):
16     print("\nEmail #{} ({})".format(idx, priority))
17     print("Subject: {}".format(subject))
18     print("Body: {}".format(body))
19
20 # Summary
21 print("\nHigh: (sum[1 for e in emails if e[2] == 'High']) | Medium: (sum[1 for e in emails if e[2] == 'Medium']) | Low: (sum[1 for e in emails if e[2] == 'Low'])")
22 print("-" * 80)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Body: The main database server is down. All operations are halted.
Email #1 (Medium Priority)
Subject: Q1 Budget Review Meeting - Next Friday at 2 PM
Body: Please review the attached budget documents.
Email #2 (Low Priority)
Subject: Office Lunch - Catering Menu For Next Week
Body: Please vote on your preferred lunch option.
Email #3 (High Priority)
Subject: Critical Client Presentation Delayed - Decision Needed Today
Body: Our major client has requested to reschedule the presentation.
Email #4 (Medium Priority)
Subject: Monthly Team Updates - Please Submit by End of Week
Body: Submit your monthly progress report by Friday.
Email #5 (Low Priority)
Subject: Office Supplies Restocking - New Printer Paper Available
Body: New printer paper has arrived in the supply closet.

High: 2 | Medium: 2 | Low: 2
PS C:\Users\parva\Desktop\AI Assisted>

OUTLINE

TIMELINE

28°C

File Edit Selection View Go Run Terminal Help

Q AI Assisted

CHAT

SENTIMENT CLASSIFICATION FOR CUST...

Write a very short Python program that stores 6 sample email messages with their subject, body and priority levels (High, Medium, Low). Using lists or tuples is fine. Keep the code simple and minimal.

Used 1 reference

```

# Simple list of email tuples: (subject, body, priority)
emails = [
    ("Urgent: System Outage - Im", "Please review the attached budget documents.", "Medium"),
    ("Q1 Budget Review Meeting -", "Submit your monthly progress report by Friday.", "Medium"),
    ("Monthly Team Updates - Pla", "New printer paper has arrived in the supply closet.", "Low")
]

# Print emails with priority
print("OFFICE EMAILS")
print("-" * 80)

for idx, (subject, body, priorit) in enumerate(emails, 1):
    print("\nEmail #{} ({})".format(idx, priorit))
    print("Subject: {}".format(subject))
    print("Body: {}".format(body))

# Summary
print("\nHigh: (sum[1 for e in em...")

```

This minimal version:

- Uses tuples for simple email storage
- Stores 6 emails with subject, body, and priority
- Prints formatted output with priority levels
- Shows summary counts at the end
- Keeps code simple and easy to understand

x ass4.py Explore and understand your code

Ask Auto

In 22, Col 163 | Spaces 4 | UTF-8 | CR LF | {} Python | 3.13.7

1:37 PM

1/20/2023

File Edit Selection View Go Run Terminal Help

EXPLORER

- AI Assisted
- AI ass1.pdf
- AI ass1.pdf
- ass4.pdf
- check leap.yarpy
- lab 4.3 word.docx
- lab 4.3 word.pdf
- lab 4.3 yarpy
- lab assignment 3.3.pdf
- lab assignment 1.4.pdf
- lab assignment 1-3.pdf
- leap.yarpy

ass 4.4.py

```

1 # Simple priority classifier function
2 def classify_priority(subject, body):
3     """Classify priority based on basic keywords"""
4     text = (subject + " " + body).lower()
5
6     if any(word in text for word in ["urgent", "critical", "immediate", "outage", "emergency"]):
7         return "High"
8     elif any(word in text for word in ["important", "meeting", "review", "deadline", "required"]):
9         return "Medium"
10    else:
11        return "Low"
12
13 # Test classifier on sample emails
14 print("PRIORITY CLASSIFIER TEST")
15 print("-" * 80)
16
17 for idx, (subject, body, actual_priority) in enumerate(emails, 1):
18     predicted = classify_priority(subject, body)
19     match = "Predicted: " + predicted + " | Actual: " + actual_priority
20     print("\nEmail #{} ({})".format(idx, match))
21     print("Subject: {}".format(subject))
22     print("Body: {}".format(body))
23     print("Predicted: (predicted) | Actual: (actual_priority)")
24
25 # Test with custom email
26 print("\nCustom Email Test")
27 custom_subject = input("\nEnter email subject: ")
28 custom_body = input("\nEnter email body: ")
29 result = classify_priority(custom_subject, custom_body)
30 print("Classified Priority: (result)")

# Test classifier on sample email
print("PRIORITY CLASSIFIER TEST")
print("-" * 80)

for idx, (subject, body, actual_) in enumerate(emails, 1):
    predicted = classify_priority(subject, body)
    match = "Predicted: " + predicted + " | Actual: " + actual_
    print("\nEmail #{} ({})".format(idx, match))
    print("Subject: {}".format(subject))
    print("Body: {}".format(body))

# Test with custom email
print("\nCustom Email Test")
custom_subject = input("Enter email subject: ")
custom_body = input("Enter email body: ")
result = classify_priority(custom_subject, custom_body)
print("Classified Priority: (result)")


```

PROBLEMS

Email #1 (Low Priority)
Subject: Office Lunch - Catering Menu For Next Week
Body: Please vote on your preferred lunch option.
Email #2 (High Priority)
Subject: Critical Client Presentation Delayed - Decision Needed Today
Body: Our major client has requested to reschedule the presentation.
Email #3 (Medium Priority)
Subject: Monthly Team Updates - Please Submit by End of Week
Body: Submit your monthly progress report by Friday.
Email #4 (Low Priority)
Subject: Office Supplies Restocking - New Printer Paper Available
Body: New printer paper has arrived in the supply closet.

High: 2 | Medium: 2 | Low: 2
PS C:\Users\parva\Desktop\AI Assisted>

OUTLINE

TIMELINE

File Edit Selection View Go Run Terminal Help

Q AI Assisted

CHAT

SENTIMENT CLASSIFICATION FOR CUST...

Write a small Python code snippet to classify an email as High, Medium, or Low Priority using only basic if-else statements. Use very simple words and minimal lines of code.

Used 1 reference

```

# ...existing code...
# Simple priority classifier function
def classify_priority(subject, body):
    """Classify email priority using basic if-else statements"""

    if any(word in subject for word in ["urgent", "critical", "immediate", "outage", "emergency"]):
        return "High"
    elif any(word in subject for word in ["important", "meeting", "review", "deadline", "required"]):
        return "Medium"
    else:
        return "Low"

# Test classifier on sample email
print("PRIORITY CLASSIFIER TEST")
print("-" * 80)

for idx, (subject, body, actual_) in enumerate(emails, 1):
    predicted = classify_priority(subject, body)
    match = "Predicted: " + predicted + " | Actual: " + actual_
    print("\nEmail #{} ({})".format(idx, match))
    print("Subject: {}".format(subject))
    print("Body: {}".format(body))

# Test with custom email
print("\nCustom Email Test")
custom_subject = input("Enter email subject: ")
custom_body = input("Enter email body: ")
result = classify_priority(custom_subject, custom_body)
print("Classified Priority: (result)")


```

x ass4.py Explore and understand your code

Ask Auto

In 54, Col 40 | Spaces 4 | UTF-8 | CR LF | {} Python | 3.13.7

```

40 # Test with custom email
41 print("\n" + "-" * 80)
42 custom_subject = input("Enter email subject: ")
43 custom_body = input("Enter email body: ")
44 result = classify_priority(custom_subject, custom_body)
45 print("classified Priority: " + result)
46
47 # ...existing code...
48
49 def classify_priority(subject, body=""):
50     text = (subject + " " + body).lower()
51     if any(k in text for k in ("server down", "urgent", "critical", "immediate", "outage", "emergency")):
52         return "High"
53     elif any(k in text for k in ("meeting", "deadline", "important", "review", "required", "asap")):
54         return "Medium"
55     else:
56         return "Low"
57
58 if __name__ == "__main__":
59     subj = input("Email subject: ").strip()
60     body = input("Email body (optional): ").strip()
61     print("Priority: ", classify_priority(subj, body))
62
63 # ...existing code...

```

Body: Please vote on your preferred lunch option.

Email #4 [High Priority]
Subject: Critical Client Presentation Delayed - Decision Needed Today
Body: Our major client has requested to reschedule the presentation.

Email #5 [Medium Priority]
Subject: Monthly Team Updates - Please Submit by End of Week
Body: Submit your monthly progress report by Friday.

Email #6 [Low Priority]
Subject: Office Supplies Restocking - New Printer Paper Available
Body: New printer paper has arrived in the supply closet.

High: 2 | Medium: 2 | Low: 2

PS C:\Users\parva\OneDrive\Desktop\AI Assisted & C:/users/parva/appData/local/Programs/Python/Python313/python.exe Python 3.13.7 (tags/v3.13.7:bbec7e, Aug 14 2023, 14:15:11) [MS v.1944 64 bit (AMD64)] on win10

Type "help", "copyright", "credits" or "license" for more information.

(C:\Users\parva\OneDrive\Desktop\AI Assisted>>> & C:/users/parva/appData/local/Programs/Python/Python313/python.exe "c:/users/parva/OneDrive/Desktop/AI Assisted/ass 4.4.py"

>>> c:\users\parva\appData\local\Programs\Python\Python313\python.exe "c:/users/parva/OneDrive/Desktop/AI Assisted/ass 4.4.py"

File "c:\users\parva\appData\local\Programs\Python\Python313\python.exe", line 1
& C:/users/parva/appData/local/Programs/Python/Python313/python.exe "c:/users/parva/OneDrive/Desktop/AI Assisted/ass 4.4.py"

SyntaxError: invalid syntax

```

67 if __name__ == "__main__":
68     subj = input("Email subject: ").strip()
69     body = input("Email body (optional): ").strip()
70     print("Priority: ", classify_priority(subj, body))
71
72 # ...existing code...
73
74 def classify_priority(text):
75     t = text.lower()
76     if any(k in t for k in ("urgent", "server down", "issue", "critical", "immediate", "outage")):
77         return "High"
78     elif any(k in t for k in ("meeting", "tomorrow", "deadline", "schedule", "reminder")):
79         return "Medium"
80     else:
81         return "Low"
82
83 # Examples
84 for e in ["Urgent issue", "Meeting tomorrow", "Greetings"]:
85     print(f"({e}) -> {classify_priority(e)}")

```

Body: Please vote on your preferred lunch option.

Email #4 [High Priority]
Subject: Critical Client Presentation Delayed - Decision Needed Today
Body: Our major client has requested to reschedule the presentation.

Email #5 [Medium Priority]
Subject: Monthly Team Updates - Please Submit by End of Week
Body: Submit your monthly progress report by Friday.

Email #6 [Low Priority]
Subject: Office Supplies Restocking - New Printer Paper Available
Body: New printer paper has arrived in the supply closet.

High: 2 | Medium: 2 | Low: 2

PS C:\Users\parva\OneDrive\Desktop\AI Assisted & C:/users/parva/appData/local/Programs/Python/Python313/python.exe Python 3.13.7 (tags/v3.13.7:bbec7e, Aug 14 2023, 14:15:11) [MS v.1944 64 bit (AMD64)] on win10

Type "help", "copyright", "credits" or "license" for more information.

(C:\Users\parva\OneDrive\Desktop\AI Assisted>>> & C:/users/parva/appData/local/Programs/Python/Python313/python.exe "c:/users/parva/OneDrive/Desktop/AI Assisted/ass 4.4.py"

>>> c:\users\parva\appData\local\Programs\Python\Python313\python.exe "c:/users/parva/OneDrive/Desktop/AI Assisted/ass 4.4.py"

File "c:\users\parva\appData\local\Programs\Python\Python313\python.exe", line 1
& C:/users/parva/appData/local/Programs/Python/Python313/python.exe "c:/users/parva/OneDrive/Desktop/AI Assisted/ass 4.4.py"

SyntaxError: invalid syntax

3. Student Query Routing System

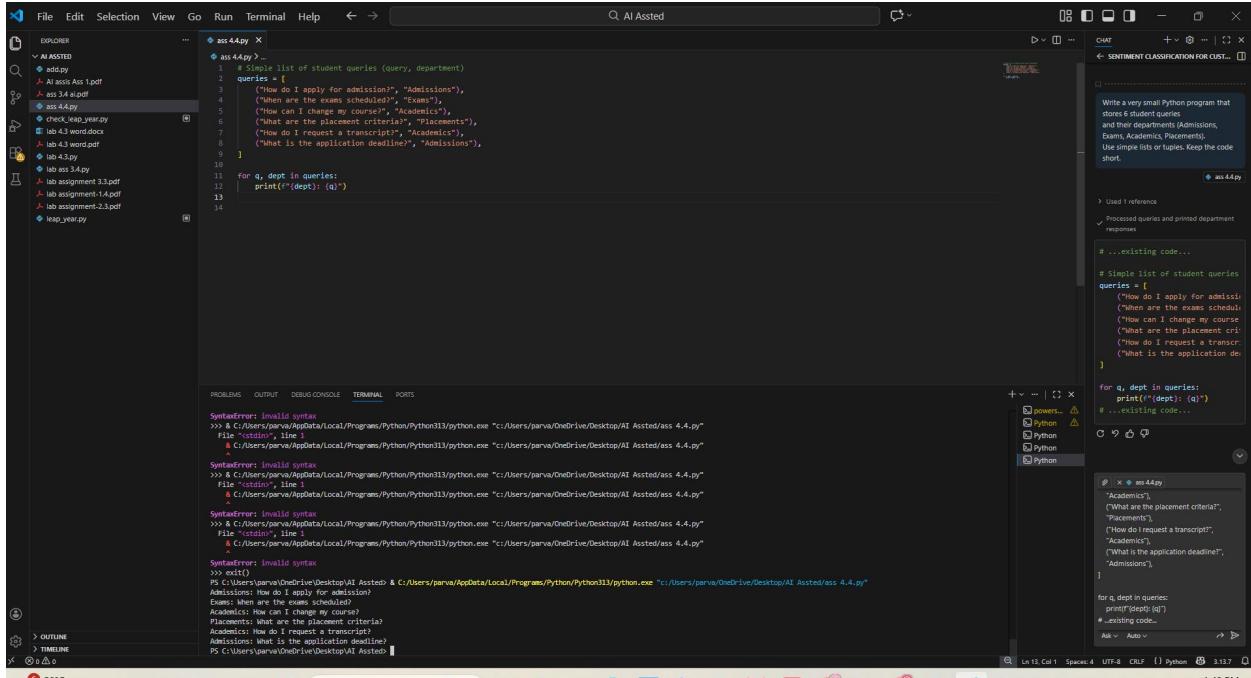
Scenario:

A university chatbot must route student queries to Admissions, Exams,

Academics, or Placements.

Tasks:

1. Create 6 sample student queries mapped to departments.
2. Implement Zero-shot intent classification using an LLM.
3. Improve results using One-shot prompting.
4. Further refine results using Few-shot prompting.
5. Analyze how contextual examples affect classification accuracy.



The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files in the "AI Assisted" folder, including "ass 4.4.py", "check_leap_year.py", "lab 3 worddocs", "lab 4.1 worddocs", "lab 4.2 worddocs", "lab 4.3 worddocs", "lab ass 3.4.py", "lab assignment 3.3.pdf", "lab assignment 1.4.pdf", "lab assignment 2.3.pdf", and "leap_year.py".
- Code Editor:** Displays the content of "ass 4.4.py". The code defines a function `print_department` that takes a query string and prints it along with its department. A comment at the top specifies that the input is a simple list of student queries (query, department) pairs.
- Terminal:** Shows command-line output from running the script. It lists several student queries and their corresponding departments, such as "Admissions", "Academics", "Placements", and "Exams".
- Output:** Shows the results of the execution, which are the processed queries and their mapped departments.
- Right-hand sidebar:** Contains sections for "CHAT", "SENTIMENT CLASSIFICATION FOR CUST...", and "Used References". It also includes a "Used References" section and a "Processed queries and printed department responses" section.
- Status Bar:** Shows the current file is "ass 4.4.py", the line number is 13, and the column number is 1. It also indicates the file is in Python mode.

```

# Simple list of student queries (query, department)
queries = [
    ("How do I apply for admissions?", "Admissions"),
    ("When are the exams scheduled?", "Exams"),
    ("How can I change my course?", "Academics"),
    ("What are the placement criteria?", "Placements"),
    ("How do I request a transcript?", "Academics"),
    ("What is the application deadline?", "Admissions"),
]
for q, dept in queries:
    print(f"{dept}: {q}")

```

The screenshot shows the AI Assisted IDE interface with a Python script named `ass_4.4.py` open in the editor. The code defines a function `def classify_query(q)` that takes a query string `q` and prints its department. The logic involves checking if the query contains specific words related to admission, exams, placements, or academic records. A sidebar on the right provides a "SENTIMENT CLASSIFICATION FOR CUSTOMER SUPPORT" example, and a bottom panel shows a terminal window with command-line interactions.

```

1 # ...existing code...
2
3 def classify_query(q):
4     t = q.lower()
5     if any(k in t for k in ("admission", "apply", "application", "enroll", "fee", "deadline")):
6         return "Admissions"
7     elif any(k in t for k in ("exam", "exams", "result", "results", "grade", "schedule")):
8         return "Exams"
9     elif any(k in t for k in ("placement", "placements", "internship", "job", "career", "interview")):
10        return "Placements"
11    elif any(k in t for k in ("course", "curriculum", "transcript", "semester", "register", "attendance", "academic")):
12        return "Academics"
13    else:
14        return "Academics"
15
16 if __name__ == "__main__":
17     q = input("Query: ").strip()
18     print("Department:", classify_query(q))
19
20 # ...existing code...

```

This screenshot shows the same AI Assisted IDE interface with a different Python script, also named `ass_4.4.py`. The code is identical to the one above, defining the `classify_query` function. The terminal window at the bottom shows a command-line session where the user runs the script and inputs a query, receiving a response from the program.

```

1 # ...existing code...
2
3 def classify_query(q):
4     t = q.lower()
5     if any(k in t for k in ("admission", "apply", "application", "enroll", "fee", "deadline")):
6         return "Admissions"
7     elif any(k in t for k in ("exam", "exams", "result", "results", "grade", "schedule")):
8         return "Exams"
9     elif any(k in t for k in ("placement", "placements", "internship", "job", "career", "interview")):
10        return "Placements"
11    elif any(k in t for k in ("course", "curriculum", "transcript", "semester", "register", "attendance", "academic")):
12        return "Academics"
13    else:
14        return "Academics"
15
16 if __name__ == "__main__":
17     q = input("Query: ").strip()
18     print("Department:", classify_query(q))
19
20 # ...existing code...

```

The screenshot shows a Windows desktop environment with a code editor and a terminal window.

Code Editor (Visual Studio Code):

- Title Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Left Sidebar (Explorer):** Shows files in the current workspace, including `AI Assisted`, `add.py`, `ass 3.4 ass.pdf`, `ass 4.4.py`, `check_leap_year.py`, `leap 4.3 wordlock`, `leap 4.3.py`, `leap 3.4.py`, `leap assignment 3.3.pdf`, `leap assignment 1.4.pdf`, `leap assignment 2.3.pdf`, and `leap year.py`.
- Code Area:** The file `ass 4.4.py` contains Python code for sentiment classification. It includes functions for zero-shot learning, few-shot learning, and a main loop for testing and scoring. The code uses various NLP libraries like `nltk` and `transformers`.
- Terminal:** The terminal window shows the command `python ass 4.4.py` being run, and the output displays the accuracy of the model, which is 0.0.
- Right Sidebar:** Shows a preview of the generated PDF and a snippet of the generated Python code.

Bottom Taskbar: Includes icons for File Explorer, Task View, Start, Search, and several pinned application icons.

4. Chatbot Question Type Detection

Scenario:

A chatbot must identify whether a user query is Informational,

Transactional, Complaint, or Feedback.

Tasks:

1. Prepare 6 chatbot queries mapped to question types.
2. Design prompts for Zero-shot, One-shot, and Few-shot learning.
3. Test all prompts on the same unseen queries.
4. Compare response correctness and ambiguity handling.
5. Document observations.

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files including `ass 4.4.pdf`, `check_leap_year.py`, `lab 4.3 word.docx`, `lab 4.3 word.pdf`, `lab 4.3.pdf`, `lab assignment 3.3.pdf`, `lab assignment 1.4.pdf`, `lab assignment 2.3.pdf`, `leap_year.py`, and `ass 4.4.py`.
- Code Editor:** Displays the `ass 4.4.py` file content. The code defines three functions: `zero_shot(a)`, `one_shot(a)`, and `few_shot(a)`. It uses regular expressions to identify patterns in the input string `a` and returns corresponding responses based on the detected words.
- Terminal:** Shows the command `>> hi`.
- Output:** Shows the output of the code execution.
- Right Panel:** Contains a sidebar titled "SENTIMENT CLASSIFICATION FOR CUSTOMERS". It lists steps and examples:
 - if else, no examples
 - b) One-shot (include one example as a comment or guide)
 - c) Few-shot (include 2-4 examples as comment or guide)
 - 3. Test all three approaches on the same unseen chatbot queries.
 - 4. Print the selected question type for each approach
 - 5. At the end, print 2-3 short lines comparing:
 - a) correctness
 - b) ambiguity handling
 - 6. Keep the code beginner-friendly and minimal.
 - 7. Do NOT use ML or AI libraries.
 - 8. Avoid extra explanations; focus on short, readable code only.

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.
3. Use One-shot prompting with an example.
4. Use Few-shot prompting with multiple emotions.
5. Discuss ambiguity handling across techniques.

```
File Edit Selection View Go Run Terminal Help < > Q AI Coding
```

EXPLORER ...

AI CODING

- add.py
- Al lab43.py
- Assignment1(CP).pdf
- Assignment 2-4.pdf
- assignment 3.4
- assignment 3.py
- Assignment2.pdf
- assignment3.4.docx
- factorial.py
- jobs.py
- jobscc.py
- lab assignment ...
- lab assignment3.3.pdf
- lab1 HCP.pdf
- matrixHCP.py
- Untitled20.ipynb
- week2 HCP.pdf

Welcome lab assignment 44.py > {} pd

Generate code

Add Context...

```
1 import pandas as pd
2
3 # Create a DataFrame from the provided data
4 data = {
5     "Text": [
6         "I am very happy today",
7         "I feel lonely and depressed",
8         "This is so frustrating",
9         "I am worried about my future",
10        "Today is just normal",
11        "Feeling excited about results"
12    ],
13    "Emotion": [
14        "Happy",
15        "Sad",
16        "Angry",
17        "Anxious",
18        "Neutral",
19        "Happy"
20    ]
21 }
22
23 df = pd.DataFrame(data)
24
25 # Display the DataFrame
26 print(df)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'.
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'.
PS D:\AI Coding>
```

```
File Edit Selection View Go Run Terminal Help < > Q AI Coding
```

EXPLORER ...

AI CODING

- add.py
- Al lab43.py
- Assignment1(CP).pdf
- Assignment 2-4.pdf
- assignment 3.4
- assignment 3.py
- Assignment2.pdf
- assignment3.4.docx
- factorial.py
- jobs.py
- jobscc.py
- lab assignment ...
- lab assignment3.3.pdf
- lab1 HCP.pdf
- matrixHCP.py
- Untitled20.ipynb
- week2 HCP.pdf

Welcome lab assignment 44.py > {} pd

Generate code

Add Context...

```
1 import pandas as pd
2
3 # Create a DataFrame from the provided data
4 data = {
5     "Text": [
6         "I am very happy today",
7         "I feel lonely and depressed",
8         "This is so frustrating",
9         "I am worried about my future",
10        "Today is just normal",
11        "Feeling excited about results"
12    ],
13    "Emotion": [
14        "Happy",
15        "Sad",
16        "Angry",
17        "Anxious",
18        "Neutral",
19        "Happy"
20    ]
21 }
22
23 df = pd.DataFrame(data)
24
25 # Display the DataFrame
26 print(df)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'.
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'.
PS D:\AI Coding>
```

The screenshot shows the Visual Studio Code interface with the 'AI Coding' workspace selected. In the center editor area, the file 'lab assignment 44.py' is open, displaying the following Python code:

```
1 def identify_emotion(text):
2     if "frustrating" in text:
3         return "Frustrated"
4     return "Neutral"
5
6 # Example usage
7 text = "This is so frustrating"
8 emotion = identify_emotion(text)
9 print(f"Emotion: {emotion}")
```

A code completion dropdown is visible above the code, showing suggestions for the word 'identify'. The dropdown title is 'Generate code' and it includes an 'Add Context...' option. The status bar at the bottom right shows 'Auto'.

The screenshot shows the Visual Studio Code interface with the 'AI Coding' workspace selected. In the center editor area, the file 'lab assignment 44.py' is open, displaying the following Python code:

```
1 def classify_emotion(text):
2     emotions = {
3         "happy": ["happy", "joyful", "excited", "pleased"],
4         "sad": ["lonely", "depressed", "sad", "down"],
5         "anxious": ["worried", "anxious", "nervous", "stressed"],
6         "neutral": ["normal", "fine", "okay", "average"],
7         "frustrated": ["frustrating", "annoyed", "irritated"]
8     }
9
10    for emotion, keywords in emotions.items():
11        if any(keyword in text.lower() for keyword in keywords):
12            return emotion
13    return "Unknown"
14
15 # Example usage
16 text = "This is so frustrating"
17 emotion = classify_emotion(text)
18 print(f"Text: \'{text}\'\nEmotion: {emotion}")
```

A code completion dropdown is visible above the code, showing suggestions for the word 'classify'. The dropdown title is 'Generate code' and it includes an 'Add Context...' option. The status bar at the bottom right shows 'Auto'.

The screenshot shows a code editor interface with the following details:

- File Explorer (EXPLORER):** Shows files in the "AI CODING" folder, including "add.py", "AI lab43.py", "Assignment1(CP).pdf", "Assignment 2-4.pdf", "assignment 3.4", "assignment 3.py", "Assignment2.pdf", "Assignment2.4.docx", "factorial.py", "jobs.py", "jobs.py", "lab assignment ...", "lab assignment3.3.pdf", "lab1 HCP.pdf", "matrixHCP.py", "Untitled20.py", and "week2 HCP.pdf".
- Code Editor (LAB ASSIGNMENT 44.py):** Displays the following Python code:

```
def classify_emotion(text):
    emotions = {
        "happy": ["happy", "joyful", "excited", "pleased"],
        "sad": ["lonely", "depressed", "sad", "down"],
        "anxious": ["worried", "anxious", "nervous", "stressed"],
        "neutral": ["normal", "fine", "okay", "average"],
        "frustrated": ["frustrating", "annoyed", "irritated"]
    }

    for emotion, keywords in emotions.items():
        if any(keyword in text.lower() for keyword in keywords):
            return emotion
    return "Unknown"

# Example usage
text = "This is so frustrating"
emotion = classify_emotion(text)
print(f"Text: '{text}'\nEmotion: {emotion}")
```
- Terminal (TERMINAL):** Shows the following command-line session:

```
Traceback (most recent call last):
  File "d:\AI Coding\lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding > C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Anxious
PS D:\AI Coding > C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Frustrated
PS D:\AI Coding > C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Text: "This is so frustrating"
Emotion: frustrated
PS D:\AI Coding>
```
- Sidebar:** Includes icons for file operations (New, Open, Save, Delete), a refresh button, and buttons for "OUTLINE" and "TIMELINE".