AI Assistant Coding

Assignment 4.1.2

Name : N. Eshwar **HT. No :** 2303A52072 **Batch:** 32

Q1.

A company receives a large number of customer emails every day and wants to automatically classify them into the following

categories:

• Billing

• Technical Support

• Feedback

• Others

Instead of training a new machine learning model, the company decides to use prompt engineering techniques with an existing large language model.

Tasks

1. Prepare five short sample emails, each belonging to one of the above categories.

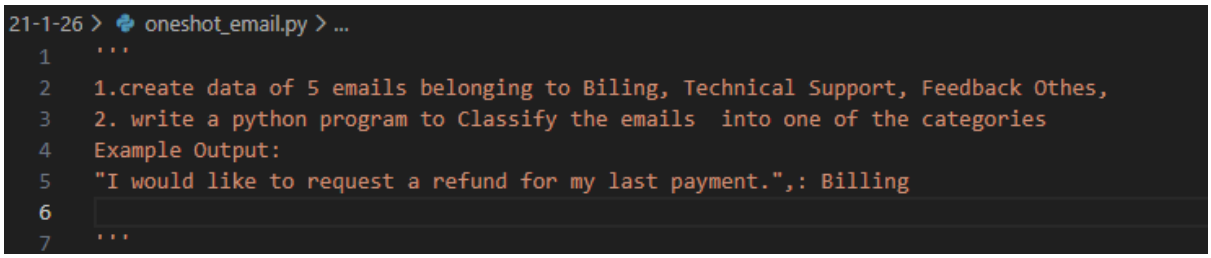
2. Write a zero-shot prompt to classify a given email into one of the categories without providing any examples.

```
import re
class EmailClassifier:
    def __init__(self):
        self.categories = {
            'Billing': re.compile(r'\b(invoice|payment|bill|charge|refund|subscription)\b', re.IGNORECASE),
            'Technical Support': re.compile(r'\b(error|issue|bug|crash|technical support|help)\b', re.IGNORECASE),
            'Feedback': re.compile(r'\b(feedback|suggestion|improvement|comment|review)\b', re.IGNORECASE)
        }

    def classify_email(self, email_content):
        for category, pattern in self.categories.items():
            if pattern.search(email_content):
                return category
        return 'Others'
# Sample emails
emails = [
    "I would like to request a refund for my last payment.",
    "The app crashes every time I try to open it.",
    "Here is some feedback on your latest update.",
    "Can you help me with an issue I'm facing?",
    "Just wanted to say thank you for the great service!"
]
classifier = EmailClassifier()
for i, email in enumerate(emails, 1):
    category = classifier.classify_email(email)
    print(f"Email {i}: {category}")
```

```
PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/Activate.ps1
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/emil.py
Email 1: Billing
Email 2: Others
Email 3: Feedback
Email 4: Technical Support
Email 5: Others
(AIAC_env) PS D:\Course\AIAC>
```

3. Write a one-shot prompt by including one labeled email example and ask the model to classify a new email.



```
21-1-26 > oneshot_email.py > ...
1   '''
2   1.create data of 5 emails belonging to Biling, Technical Support, Feedback Othes,
3   2. write a python program to Classify the emails  into one of the categories
4   Example Output:
5   "I would like to request a refund for my last payment.",: Billing
6
7   '''
```

```
import re
class EmailClassifier:
    def __init__(self):
        self.categories = {
            'Billing': re.compile(r'\b(invoice|payment|bill|charge|refund|subscription)\b', re.IGNORECASE),
            'Technical Support': re.compile(r'\b(error|issue|bug|crash|technical support|help)\b', re.IGNORECASE),
            'Feedback': re.compile(r'\b(feedback|suggestion|improvement|comment|review)\b', re.IGNORECASE)
        }

    def classify_email(self, email_content):
        for category, pattern in self.categories.items():
            if pattern.search(email_content):
                return category
        return 'Others'
# Sample emails
emails = [
    "I would like to request a refund for my last payment.",
    "The app crashes every time I try to open it.",
    "Here is some feedback on your latest update.",
    "Can you help me with an issue I'm facing?",
    "Just wanted to say thank you for the great service!"
]
classifier = EmailClassifier()
for i, email in enumerate(emails, 1):
    category = classifier.classify_email(email)
    print(f"Email {i}: {category}")
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/oneshot_email.py
Email 1: Billing
Email 2: Others
Email 3: Feedback
Email 4: Technical Support
Email 5: Others
(AIAC_env) PS D:\Course\AIAC>
```

4. Write a few-shot prompt by including two or three labelled email examples and ask the model to classify a new email.

```python
import re
class EmailClassifier:
    def __init__(self):
        self.categories = {
            'Billing': re.compile(r'\b(invoice|payment|bill|charge|refund|subscription)\b', re.IGNORECASE),
            'Technical Support': re.compile(r'\b(error|issue|bug|crash|technical support|help)\b', re.IGNORECASE),
            'Feedback': re.compile(r'\b(feedback|suggestion|improvement|comment|review)\b', re.IGNORECASE)
        }

    def classify_email(self, email_content):
        for category, pattern in self.categories.items():
            if pattern.search(email_content):
                return category
        return 'Others'
# Sample emails
emails = [
    "I would like to request a refund for my last payment.",
    "The app crashes every time I try to open it.",
    "Here is some feedback on your latest update.",
    "Can you help me with an issue I'm facing?",
    "Just wanted to say thank you for the great service!"
]
classifier = EmailClassifier()
for i, email in enumerate(emails, 1):
    category = classifier.classify_email(email)
    print(f"Email {i}: {category}")
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/oneshot_email.py
Email 1: Billing
Email 2: Others
Email 3: Feedback
Email 4: Technical Support
Email 5: Others
(AIAC_env) PS D:\Course\AIAC>
```

5. Compare the outputs obtained using zero-shot, one-shot, and few-shot prompting techniques and briefly comment on their effectiveness

Using zero Shot,one shot, few shot there is no changes in code just a slight improve

---

Q2.

A company wants to deploy a chatbot to handle customer queries. Each query must be classified into one of the following intents: Account Issue, Order Status, Product Inquiry, or General Question using prompt engineering techniques.

Tasks to be Completed

1. Prepare Sample Data Create 6 short chatbot user queries, each mapped to one of the four intents.

2. Zero-shot Prompting Design a prompt that asks the LLM to classify a user query into the given intent categories without examples.

```
'''
1.Create a  sample data of 10 queries belonging to Account Issues, Order status, Product Inquiry , Or General Questions
2. Write a python program to classify the queries into one of the categories Account Issues, Order status, Product Inquiry , Or General Questions
'''
```

```python
import re
class QueryClassifier:
    def __init__(self):
        self.categories = {
            'Account Issues': re.compile(r'\b(account|login|password|access|profile|settings)\b', re.IGNORECASE),
            'Order Status': re.compile(r'\b(order|shipping|delivery|status|tracking|purchase)\b', re.IGNORECASE),
            'Product Inquiry': re.compile(r'\b(product|item|specification|feature|availability|price)\b', re.IGNORECASE)
        }

    def classify_query(self, query_content):
        for category, pattern in self.categories.items():
            if pattern.search(query_content):
                return category
        return 'General Questions'
# Sample queries
queries = [
    "I can't access my account, can you help?",
    "What is the status of my recent order?",
    "Can you provide more details about the product features?",
    "How do I change my account settings?",
    "What are your business hours?",
    "I forgot my password, how can I reset it?",
    "When will my order be delivered?",
    "Is this product available in different colors?",
    "How do I update my profile information?",
    "Can you tell me about your return policy?"
]
classifier = QueryClassifier()
for query in queries:
    category = classifier.classify_query(query)
    print(f'"{query}": {category}')
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/quries_zero.py
"I can't access my account, can you help?": Account Issues
"What is the status of my recent order?": Order Status
"Can you provide more details about the product features?": Product Inquiry
"How do I change my account settings?": Account Issues
"What are your business hours?": General Questions
"I forgot my password, how can I reset it?": Account Issues
"When will my order be delivered?": Order Status
"Is this product available in different colors?": Product Inquiry
"How do I update my profile information?": Account Issues
"Can you tell me about your return policy?": General Questions
```

3. One-shot Prompting Provide one labeled query in the prompt before classifying a new query.

```
'''
2. Write a python program to classify the queries into one of the categories Account Issues, Order status, Product Inquiry , Or General Questions
Example Input:
"I can't access my account, can you help?" : Account Issues
'''
```

```python
import re
class QueryClassifier:
    def __init__(self):
        self.categories = {
            'Account Issues': re.compile(r'\b(account|login|password|access|profile|settings)\b', re.IGNORECASE),
            'Order Status': re.compile(r'\b(order|shipping|delivery|status|tracking|purchase)\b', re.IGNORECASE),
            'Product Inquiry': re.compile(r'\b(product|item|specification|feature|availability|price)\b', re.IGNORECASE)
        }

    def classify_query(self, query_content):
        for category, pattern in self.categories.items():
            if pattern.search(query_content):
                return category
        return 'General Questions'
# Sample queries
queries = [
    "I can't access my account, can you help?",
    "What is the status of my recent order?",
    "Can you provide more details about the product features?",
    "How do I change my account settings?",
    "What are your business hours?",
    "I forgot my password, how can I reset it?",
    "When will my order be delivered?",
    "Is this product available in different colors?",
    "How do I update my profile information?",
    "Can you tell me about your return policy?"
]
classifier = QueryClassifier()
for query in queries:
    category = classifier.classify_query(query)
    print(f'"{query}": {category}')
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/quries_zero.py
"I can't access my account, can you help?": Account Issues
"What is the status of my recent order?": Order Status
"Can you provide more details about the product features?": Product Inquiry
"How do I change my account settings?": Account Issues
"What are your business hours?": General Questions
"I forgot my password, how can I reset it?": Account Issues
"When will my order be delivered?": Order Status
"Is this product available in different colors?": Product Inquiry
"How do I update my profile information?": Account Issues
"Can you tell me about your return policy?": General Questions
```

4. Few-shot Prompting Include 3–5 labeled intent examples to guide the LLM before classifying a new query.

```
'''
Write a python program to classify the queries into one of the categories Account Issues, Order status, Product Inquiry , Or General Questions
Example Input:
"I can't access my account, can you help?" : Account Issues
"What is the status of my recent order?": Order Status
'''
```

```python
import re
class QueryClassifier:
    def __init__(self):
        self.categories = {
            'Account Issues': re.compile(r'\b(account|login|password|access|profile|settings)\b', re.IGNORECASE),
            'Order Status': re.compile(r'\b(order|shipping|delivery|status|tracking|purchase)\b', re.IGNORECASE),
            'Product Inquiry': re.compile(r'\b(product|item|specification|feature|availability|price)\b', re.IGNORECASE)
        }

    def classify_query(self, query_content):
        for category, pattern in self.categories.items():
            if pattern.search(query_content):
                return category
        return 'General Questions'
# Sample queries
queries = [
    "I can't access my account, can you help?",
    "What is the status of my recent order?",
    "Can you provide more details about the product features?",
    "How do I change my account settings?",
    "What are your business hours?",
    "I forgot my password, how can I reset it?",
    "When will my order be delivered?",
    "Is this product available in different colors?",
    "How do I update my profile information?",
    "Can you tell me about your return policy?"
]
classifier = QueryClassifier()
for query in queries:
    category = classifier.classify_query(query)
    print(f'"{query}": {category}')
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/quries_zero.py
"I can't access my account, can you help?": Account Issues
"What is the status of my recent order?": Order Status
"Can you provide more details about the product features?": Product Inquiry
"How do I change my account settings?": Account Issues
"What are your business hours?": General Questions
"I forgot my password, how can I reset it?": Account Issues
"When will my order be delivered?": Order Status
"Is this product available in different colors?": Product Inquiry
"How do I update my profile information?": Account Issues
"Can you tell me about your return policy?": General Questions
```

5. Evaluation Apply all three techniques to the same set of test queries and document differences in performance.

For all the test cases the out put is same as there is no change in code

---

Q3.

A university collects student feedback and wants to categorize comments as Positive, Negative, or Neutral.

Questions:

a) Write a Zero-shot prompt to classify feedback sentiment.

```
'''
write a program to collect student feedback and classify the comments into positive, negative, and neutral categories.
'''
```

```python
def classify_feedback(feedback):
    positive_keywords = ['good', 'great', 'excellent', 'amazing', 'fantastic', 'love', 'like', 'satisfied']
    negative_keywords = ['bad', 'poor', 'terrible', 'hate', 'dislike', 'unsatisfied', 'awful']

    feedback_lower = feedback.lower()

    if any(word in feedback_lower for word in positive_keywords):
        return 'Positive'
    elif any(word in feedback_lower for word in negative_keywords):
        return 'Negative'
    else:
        return 'Neutral'
def collect_feedback():
    feedback_list = []
    print("Enter student feedback (type 'exit' to finish):")

    while True:
        feedback = input("Feedback: ")
        if feedback.lower() == 'exit':
            break
        feedback_list.append(feedback)

    return feedback_list
def main():
    feedbacks = collect_feedback()
    classified_feedback = {'Positive': [], 'Negative': [], 'Neutral': []}

    for feedback in feedbacks:
        category = classify_feedback(feedback)
        classified_feedback[category].append(feedback)

    print("\nClassified Feedback:")
    for category, comments in classified_feedback.items():
        print(f"\n{category} Comments:")
        for comment in comments:
            print(f"- {comment}")
if __name__ == "__main__":
    main()
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/stundent_feedback.py
Enter student feedback (type 'exit' to finish):
Feedback: The campus facilities are excellent and well maintained.
Feedback: The syllabus is outdated and not relevant to current industry needs.
Feedback: The classes are okay, but nothing special.
Feedback: exit

Classified Feedback:

Positive Comments:
- The campus facilities are excellent and well maintained.

Negative Comments:

Neutral Comments:
- The syllabus is outdated and not relevant to current industry needs.
- The classes are okay, but nothing special.
(AIAC_env) PS D:\Course\AIAC>
```

b) Provide a One-shot prompt with one feedback example.

```
'''
collect student feedback and classify the comments into positive, negative, and neutral categories.
Feedback: "The professor explained concepts clearly and was very supportive."
Sentiment: Positive
'''
```

```python
def classify_feedback(feedback):
    positive_keywords = ['good', 'great', 'excellent', 'amazing', 'fantastic', 'love', 'like', 'satisfied']
    negative_keywords = ['bad', 'poor', 'terrible', 'hate', 'dislike', 'unsatisfied', 'awful']

    feedback_lower = feedback.lower()

    if any(word in feedback_lower for word in positive_keywords):
        return 'Positive'
    elif any(word in feedback_lower for word in negative_keywords):
        return 'Negative'
    else:
        return 'Neutral'
def collect_feedback():
    feedback_list = []
    print("Enter student feedback (type 'exit' to finish):")

    while True:
        feedback = input("Feedback: ")
        if feedback.lower() == 'exit':
            break
        feedback_list.append(feedback)

    return feedback_list

def main():
    feedbacks = collect_feedback()
    classified_feedback = {'Positive': [], 'Negative': [], 'Neutral': []}

    for feedback in feedbacks:
        category = classify_feedback(feedback)
        classified_feedback[category].append(feedback)

    print("\nClassified Feedback:")
    for category, comments in classified_feedback.items():
        print(f"\n{category} Comments:")
        for comment in comments:
            print(f"- {comment}")
if __name__ == "__main__":
    main()
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/student_feedback_one.py
Enter student feedback (type 'exit' to finish):
Feedback: The campus facilities are excellent and well maintained.
Feedback: The syllabus is outdated and not relevant to current industry needs.
Feedback: The classes are okay, but nothing special.
Feedback: exit

Classified Feedback:

Positive Comments:
- The campus facilities are excellent and well maintained.

Negative Comments:

Neutral Comments:
- The syllabus is outdated and not relevant to current industry needs.
- The classes are okay, but nothing special.
(AIAC_env) PS D:\Course\AIAC>
```

c) Create a Few-shot prompt using multiple labeled feedback samples.

```
write a python progarm to collect feedback from students about university and classify those comments into positive negative and nutral:
Example Input:
Feedback: "The campus facilities are excellent and well maintained."
Sentiment: Positive

Feedback: "The syllabus is outdated and not relevant to current industry needs."
Sentiment: Negative

Feedback: "The classes are okay, but nothing special."
Sentiment: Neutral
```

```python
def classify_feedback(feedback: str) -> str:
    text = feedback.lower()
    positive_keywords = {"excellent", "great", "good", "amazing", "well maintained", "love", "helpful", "friendly", "awesome", "satisfied"}
    negative_keywords = {"bad", "poor", "outdated", "irrelevant", "not relevant", "boring", "confusing", "hate", "terrible", "disappointed"}

    if any(word in text for word in positive_keywords):
        return "Positive"
    if any(word in text for word in negative_keywords):
        return "Negative"
    return "Neutral"


def main():
    print("Enter feedback (press Enter on an empty line to finish):")
    while True:
        feedback = input("Feedback: ").strip()
        if not feedback:
            break
        sentiment = classify_feedback(feedback)
        print(f"Sentiment: {sentiment}\n")


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

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/studen_few.py
Enter feedback (press Enter on an empty line to finish):
Feedback: The campus facilities are excellent and well maintained.
Sentiment: Positive

Feedback: The syllabus is outdated and not relevant to current industry needs.
Sentiment: Negative

Feedback: The classes are okay, but nothing special.
Sentiment: Neutral
```

d) Explain how examples improve sentiment classification accuracy.

The Examples help the AI to add more keywords in code to classify the queries

---

Q4.

An online learning platform wants to recommend courses by classifying learner queries into Beginner, Intermediate, or Advanced levels.

Questions:

a) Write a Zero-shot prompt to classify learner queries.

```
'''
1.Genarte students lerrenr queries
2. write python function to classify the queries into Bigginer, Intermediate, Advanced levels  and recommend the courses accordingly.
'''
```

```python
def classify_query(query):
    beginner_keywords = ['beginner', 'basic', 'introduction', 'getting started', 'fundamentals']
    intermediate_keywords = ['intermediate', 'project', 'application', 'development', 'implementation']
    advanced_keywords = ['advanced', 'expert', 'optimization', 'scaling', 'architecture']

    query_lower = query.lower()

    if any(keyword in query_lower for keyword in beginner_keywords):
        return "Beginner Level: Recommended Course - 'Introduction to Programming'"
    elif any(keyword in query_lower for keyword in intermediate_keywords):
        return "Intermediate Level: Recommended Course - 'Intermediate Programming Projects'"
    elif any(keyword in query_lower for keyword in advanced_keywords):
        return "Advanced Level: Recommended Course - 'Advanced Programming Techniques'"
    else:
        return "Query not classified. Please provide more details."
# Example usage
student_query = "I want to learn the basics of programming."
recommendation = classify_query(student_query)
print(recommendation)
```

```
PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/Activate.ps1
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/course_zero.py
Beginner Level: Recommended Course - 'Introduction to Programming'
(AIAC_env) PS D:\Course\AIAC>
```

b) Create a One-shot prompt with one example query.

```
'''
1. Generate student learner 0 queries
2. Write a Python function to classify the queries into Beginner, Intermediate, Advanced levels and recommend the courses accordingly.
example queries:
Query: "what is HTML and how do I create my first web page?"
Level: Beginner
'''
```

```python
def classify_query(query):
    beginner_keywords = ['beginner', 'basic', 'introduction', 'getting started', 'fundamentals', 'html', 'css', 'first web page']
    intermediate_keywords = ['intermediate', 'project', 'application', 'development', 'implementation', 'javascript', 'responsive design']
    advanced_keywords = ['advanced', 'expert', 'optimization', 'scaling', 'architecture', 'performance tuning', 'security']

    query_lower = query.lower()

    if any(keyword in query_lower for keyword in beginner_keywords):
        return "Beginner Level: Recommended Course - 'Introduction to Web Development'"
    elif any(keyword in query_lower for keyword in intermediate_keywords):
        return "Intermediate Level: Recommended Course - 'Intermediate Web Development Projects'"
    elif any(keyword in query_lower for keyword in advanced_keywords):
        return "Advanced Level: Recommended Course - 'Advanced Web Development Techniques'"
    else:
        return "Query not classified. Please provide more details."
# Example usage
student_query1 = "I want to learn the basics of web development."
recommendation1 = classify_query(student_query1)
print(recommendation1)
student_query2 = "How can I optimize my website for better performance?"
recommendation2 = classify_query(student_query2)
print(recommendation2)
student_query3 = "What are some good projects for learning JavaScript?"
recommendation3 = classify_query(student_query3)
print(recommendation3)
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/corse_one.py
Beginner Level: Recommended Course - 'Introduction to Web Development'
Query not classified. Please provide more details.
Intermediate Level: Recommended Course - 'Intermediate Web Development Projects'
(AIAC_env) PS D:\Course\AIAC>
```

c) Develop a Few-shot prompt with multiple labeled queries.

```
'''
1. Generate list of student learner queries.
2. Write a Python function to classify the queries into Beginner, Intermediate, Advanced levels and recommend the courses accordingly.
Exaple queries:
Query: "Can you help me understand the fundamentals of Python programming?"
Level: Beginner
Query: "What are some good intermediate projects to practice my coding skills?"
Level: Intermediate
Query: "How do I implement advanced algorithms in my applications?"
Level: Advanced
Query: "I want to learn about data structures and algorithms."
Level: Beginner
'''
```

```
'''
def classify_query(query):
    beginner_keywords = ['beginner', 'basic', 'introduction', 'getting started', 'fundamentals', 'data structures', 'algorithms']
    intermediate_keywords = ['intermediate', 'project', 'application', 'development', 'implementation', 'coding skills']
    advanced_keywords = ['advanced', 'expert', 'optimization', 'scaling', 'architecture', 'advanced algorithms']

    query_lower = query.lower()

    if any(keyword in query_lower for keyword in beginner_keywords):
        return "Beginner Level: Recommended Course - 'Introduction to Python Programming'"
    elif any(keyword in query_lower for keyword in intermediate_keywords):
        return "Intermediate Level: Recommended Course - 'Intermediate Python Projects'"
    elif any(keyword in query_lower for keyword in advanced_keywords):
        return "Advanced Level: Recommended Course - 'Advanced Python Techniques'"
    else:
        return "Query not classified. Please provide more details."
# Example usage
student_query1 = "Can you help me understand the fundamentals of Python programming?"
recommendation1 = classify_query(student_query1)

print(recommendation1)
student_query2 = "What are some good intermediate projects to practice my coding skills?"
recommendation2 = classify_query(student_query2)

print(recommendation2)
student_query3 = "How do I implement advanced algorithms in my applications?"
recommendation3 = classify_query(student_query3)

print(recommendation3)
student_query4 = "I want to learn about data structures and algorithms."
recommendation4 = classify_query(student_query4)

print(recommendation4)
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/course_few.py
Beginner Level: Recommended Course - 'Introduction to Python Programming'
Intermediate Level: Recommended Course - 'Intermediate Python Projects'
Beginner Level: Recommended Course - 'Introduction to Python Programming'
Beginner Level: Recommended Course - 'Introduction to Python Programming'
(AIAC_env) PS D:\Course\AIAC>
```

d) Discuss how Few-shot prompting improves recommendation quality

Few Shot Prompting to help the AI to add more keywords which help to classify the queries

---

Q5. A social media platform wants to classify posts into Acceptable, Offensive, or Spam.

Questions:

a) Write a Zero-shot prompt for post moderation.

```
 1.generate some social media posts
 @.write a python program to classfy the social media posts into Acceptable, Offensie, Or spam
 '''
```

```python
import re
def classify_post(post):
    offensive_keywords = ['hate', 'stupid', 'idiot', 'dumb']
    spam_keywords = ['buy now', 'click here', 'subscribe', 'free', 'limited time offer']

    post_lower = post.lower()

    # Check for offensive content
    for word in offensive_keywords:
        if re.search(r'\b' + re.escape(word) + r'\b', post_lower):
            return 'Offensive'

    # Check for spam content
    for phrase in spam_keywords:
        if phrase in post_lower:
            return 'Spam'

    return 'Acceptable'
# Example usage
posts = [
    "I hate when people are late!",
    "Click here to get a free gift!",
    "What a beautiful day!",
    "You are so stupid!",
    "Subscribe to our newsletter for more updates."
]

for post in posts:
    classification = classify_post(post)
    print(f"Post: {post}\nClassification: {classification}\n")
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/scila_zero.py
Post: I hate when people are late!
Classification: Offensive

Post: Click here to get a free gift!
Classification: Spam

Post: What a beautiful day!
Classification: Acceptable

Post: You are so stupid!
Classification: Offensive

Post: Subscribe to our newsletter for more updates.
Classification: Spam
```

b) Convert it into a One-shot prompt.

```
'''
1. generate some social media posts
2. write a python program to classify the social media posts into Acceptable, Offensive, Or Spam
Example social media posts:
"Win a free iPhone by clicking this link now!"
→ Spam
'''
```

```python
import re
def classify_post(post):
    offensive_keywords = ['hate', 'stupid', 'idiot', 'dumb']
    spam_keywords = ['buy now', 'click here', 'subscribe', 'free', 'limited time offer']

    post_lower = post.lower()

    # Check for offensive content
    for word in offensive_keywords:
        if re.search(r'\b' + re.escape(word) + r'\b', post_lower):
            return 'Offensive'

    # Check for spam content
    for phrase in spam_keywords:
        if phrase in post_lower:
            return 'Spam'

    return 'Acceptable'
# Example usage
posts = [
    "I hate when people are late!",
    "Click here to get a free gift!",
    "What a beautiful day!",
    "You are so stupid!",
    "Subscribe to our newsletter for more updates."
]

for post in posts:
    classification = classify_post(post)
    print(f"Post: {post}\nClassification: {classification}\n")
```

```
PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/Activate.ps1
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/socil_one.py
Post: I hate when people are late!
Classification: Offensive

Post: Click here to get a free gift!
Classification: Spam

Post: What a beautiful day!
Classification: Acceptable

Post: You are so stupid!
Classification: Offensive

Post: Subscribe to our newsletter for more updates.
Classification: Spam
```

c) Design a Few-shot prompt using multiple examples.

```
'''
1. generate some social media posts
2. write a python program to classify the social media posts into Acceptable, Offensive, Or Spam
Example social media posts:
"Win a free iPhone by clicking this link now!"
→ Spam

"You are stupid and your opinion is worthless."
→ Offensive

"Had a great day at the park with friends."
→ Acceptable

"Visit this website to earn money fast with no effort!"
```

```python
import re
def classify_post(post):
    offensive_keywords = ['hate', 'stupid', 'idiot', 'dumb']
    spam_keywords = ['buy now', 'click here', 'subscribe', 'free', 'limited time offer']

    post_lower = post.lower()

    # Check for offensive content
    for word in offensive_keywords:
        if re.search(r'\b' + re.escape(word) + r'\b', post_lower):
            return 'Offensive'

    # Check for spam content
    for phrase in spam_keywords:
        if phrase in post_lower:
            return 'Spam'

    return 'Acceptable'
# Example usage
posts = [
    "I hate when people are late!",
    "Click here to get a free gift!",
    "What a beautiful day!",
    "You are so stupid!",
    "Subscribe to our newsletter for more updates."
]
for post in posts:
    classification = classify_post(post)
    print(f"Post: {post}\nClassification: {classification}\n")
```

```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/21-1-26/social_few.py
Post: I hate when people are late!
Classification: Offensive

Post: Click here to get a free gift!
Classification: Spam

Post: What a beautiful day!
Classification: Acceptable

Post: You are so stupid!
Classification: Offensive

Post: Subscribe to our newsletter for more updates.
Classification: Spam

(AIAC_env) PS D:\Course\AIAC>
```

d) Explain the challenges of Zero-shot prompting in content moderation.

Limitations in keywords which cant classify all the pots