LAB ASSIGNMENT-4.4

P.CHITHRA

2303A52137

Batch – 41

Question 1: Sentiment Classification for Customer Reviews

Task 1: Zero-Shot Prompt

Generate a code to analyze customer reviews and classify them into exactly one of these labels: Positive, Negative or Neutral sentiments. Generate 10 customer reviews. Generate a efficient and readable code. Generate in one block. No need to print the generated reviews in the output.

CODE:

```python
# First, let's create a list of 10 sample customer reviews that we will use for sentiment analysis.
customer_reviews = [
    "This product is absolutely amazing! I love it.",
    "The service was terrible and I'm very disappointed.",
    "It's an okay product, nothing special.",
    "I had a great experience with their support team.",
    "The quality of the item is very poor for the price.",
    "Fast shipping and the product arrived in perfect condition.",
    "I'm indifferent about this, it just works.",
    "Highly recommend! Will definitely buy again.",
    "Worst purchase ever. Complete waste of money.",
    "The documentation is a bit confusing, but I managed."
]

# The user requested to remove the printing of the generated reviews.
# for i, review in enumerate(customer_reviews):
#     print(f"Review {i+1}: {review}")

# Now, we will use a pre-trained sentiment analysis model from the `transformers` library to classify these reviews. We'll use a `pipeline` for easy implementation.

# Install the transformers library if you haven't already
# !pip install transformers

from transformers import pipeline

# Load a pre-trained sentiment analysis model
# 'distilbert-base-uncased-finetuned-sst-2-english' is a good general-purpose sentiment model.
sentiment_pipeline = pipeline('sentiment-analysis')
```

```python
    result = sentiment_pipeline(text)[0]
    label = result['label']
    score = result['score']

    # Map labels to 'Positive', 'Negative', 'Neutral'
    # The chosen model typically outputs 'POSITIVE' or 'NEGATIVE'.
    # We'll need a custom logic for 'Neutral' if the model doesn't directly provide it.
    # For this specific model, we can interpret low confidence in POSITIVE/NEGATIVE as Neutral.
    if label == 'POSITIVE' and score > 0.9:
        return 'Positive'
    elif label == 'NEGATIVE' and score > 0.9:
        return 'Negative'
    else:
        # For reviews where the model is not highly confident in positive or negative,
        # or if the model's score for its primary label is below a threshold, we can consider it neutral.
        # This threshold can be adjusted based on desired strictness.
        # A more robust solution might use a model specifically trained for 3-class sentiment.
        return 'Neutral'

# Classify each review and store the results
sentiment_results = []
for review in customer_reviews:
    sentiment = classify_sentiment(review)
    sentiment_results.append({'review': review, 'sentiment': sentiment})

# Display the results
print("\nSentiment Analysis Results:")
for i, result in enumerate(sentiment_results):
    print(f"Review {i+1}: {result['review']} -> Sentiment: {result['sentiment']}")
```

OUTPUT:

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingface.co/distilbert/distilbert-base-uncas
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cpu

Sentiment Analysis Results:
Review 1: This product is absolutely amazing! I love it. -> Sentiment: Positive
Review 2: The service was terrible and I'm very disappointed. -> Sentiment: Negative
Review 3: It's an okay product, nothing special. -> Sentiment: Neutral
Review 4: I had a great experience with their support team. -> Sentiment: Positive
Review 5: The quality of the item is very poor for the price. -> Sentiment: Negative
Review 6: Fast shipping and the product arrived in perfect condition. -> Sentiment: Positive
Review 7: I'm indifferent about this, it just works. -> Sentiment: Positive
Review 8: Highly recommend! Will definitely buy again. -> Sentiment: Positive
Review 9: Worst purchase ever. Complete waste of money. -> Sentiment: Negative
Review 10: The documentation is a bit confusing, but I managed. -> Sentiment: Positive
```

Task 2: One-Shot Prompt:

Analyze the review and the classify them into positive ,negative or neutral sentiments :
Example - Input-Customer Review: The product quality is excellent and I am very satisfied.
;Output- Sentiment: Positive. Now generate a efficient and readable code for 10 customer
reviews classifying into sentiments.

CODE:

```python
# Install the transformers library if you haven't already
# !pip install transformers

from transformers import pipeline

# Load a pre-trained sentiment analysis model
# 'distilbert-base-uncased-finetuned-sst-2-english' is a good general-purpose sentiment model.
sentiment_pipeline = pipeline('sentiment-analysis')

# Function to classify sentiment
def classify_sentiment(text):
    result = sentiment_pipeline(text)[0]
    label = result['label']
    score = result['score']

    # Map labels to 'Positive', 'Negative', 'Neutral'
    # The chosen model typically outputs 'POSITIVE' or 'NEGATIVE'.
    # We use a score threshold to interpret low confidence as 'Neutral'.
    if label == 'POSITIVE' and score > 0.9:
        return 'Positive'
    elif label == 'NEGATIVE' and score > 0.9:
        return 'Negative'
    else:
        return 'Neutral'

# Generate 10 customer reviews
new_customer_reviews = [
    "This new feature is incredibly useful and well-implemented.",
    "I'm very disappointed with the recent update, it broke several functionalities.",
```

```python
# Generate 10 customer reviews
new_customer_reviews = [
    "This new feature is incredibly useful and well-implemented.",
    "I'm very disappointed with the recent update, it broke several functionalities.",
    "The product arrived quickly, but it's just average.",
    "Their customer support is outstanding, very helpful and responsive.",
    "The material quality feels cheap for the price paid.",
    "Installation was a breeze, everything worked right out of the box.",
    "I don't have strong feelings either way, it gets the job done.",
    "Absolutely love this! It exceeded my expectations.",
    "Worst experience ever, I regret this purchase.",
    "The interface is a bit clunky, but I can manage."
]

print("Generated Customer Reviews:")
for i, review in enumerate(new_customer_reviews):
    print(f"Review {i+1}: {review}")

# Classify each new review and store the results
new_sentiment_results = []
for review in new_customer_reviews:
    sentiment = classify_sentiment(review)
    new_sentiment_results.append({'review': review, 'sentiment': sentiment})

print("\nSentiment Analysis Results for New Reviews:")
for i, result in enumerate(new_sentiment_results):
    print(f"Review {i+1}: {result['review']} -> Sentiment: {result['sentiment']}")
```

OUTPUT:

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingface.
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cpu
Generated Customer Reviews:
Review 1: This new feature is incredibly useful and well-implemented.
Review 2: I'm very disappointed with the recent update, it broke several functionalities.
Review 3: The product arrived quickly, but it's just average.
Review 4: Their customer support is outstanding, very helpful and responsive.
Review 5: The material quality feels cheap for the price paid.
Review 6: Installation was a breeze, everything worked right out of the box.
Review 7: I don't have strong feelings either way, it gets the job done.
Review 8: Absolutely love this! It exceeded my expectations.
Review 9: Worst experience ever, I regret this purchase.
Review 10: The interface is a bit clunky, but I can manage.

Sentiment Analysis Results for New Reviews:
Review 1: This new feature is incredibly useful and well-implemented. -> Sentiment: Positive
Review 2: I'm very disappointed with the recent update, it broke several functionalities. -> Sentiment: Negative
Review 3: The product arrived quickly, but it's just average. -> Sentiment: Positive
Review 4: Their customer support is outstanding, very helpful and responsive. -> Sentiment: Positive
Review 5: The material quality feels cheap for the price paid. -> Sentiment: Negative
Review 6: Installation was a breeze, everything worked right out of the box. -> Sentiment: Positive
Review 7: I don't have strong feelings either way, it gets the job done. -> Sentiment: Negative
Review 8: Absolutely love this! It exceeded my expectations. -> Sentiment: Positive
Review 9: Worst experience ever, I regret this purchase. -> Sentiment: Negative
Review 10: The interface is a bit clunky, but I can manage. -> Sentiment: Positive
```

JUSTIFICATION:

Zero-shot prompting classifies without examples, so results can be inconsistent. One-shot prompting uses one example, improving clarity and accuracy. Neutral sentiment is better detected in one-shot than in zero-shot. One-shot reduces ambiguity between similar sentiments. Therefore, one-shot provides more reliable outputs than zero-shot.

Question 2: Email Priority Classification

Task 1: One-Shot Prompt

Generate a code for email priority classification. The code should prioritize incoming emails into High Priority, Medium Priority or Low Priority. Example: Input- The production server is down. Customers are unable to place orders. Output- High; Now analyze the example and generate 10 incoming emails and prioritize them into labels. Generate a efficient and readable code.

CODE:

```python
# Function to classify email priority
def classify_email_priority(email_subject_body):
    text = email_subject_body.lower()

    # High Priority Keywords
    high_priority_keywords = [
        "server down", "production outage", "critical issue", "urgent",
        "security breach", "data loss", "customers unable", "payment failed",
        "system error", "cannot access", "firewall alert"
    ]
    for keyword in high_priority_keywords:
        if keyword in text:
            return "High Priority"

    # Medium Priority Keywords
    medium_priority_keywords = [
        "bug report", "feature request", "follow up", "meeting reminder",
        "update required", "performance issue", "access request", "question about"
    ]
    for keyword in medium_priority_keywords:
        if keyword in text:
            return "Medium Priority"

    # Low Priority (default if no other keywords match)
    return "Low Priority"

# Example from the user:
example_email = "The production server is down. Customers are unable to place orders."
example_priority = classify_email_priority(example_email)
```

```python
# Example from the user:
example_email = "The production server is down. Customers are unable to place orders."
example_priority = classify_email_priority(example_email)
print(f"Input- Customer Review: {example_email}\nOutput- {example_priority}\n")

# Generate 10 sample incoming emails
incoming_emails = [
    "URGENT: Production database offline, website not loading!",
    "Request for new feature in dashboard",
    "Weekly newsletter - Latest updates and promotions",
    "Server maintenance scheduled for next month",
    "Customer reported bug in login process - intermittent issue",
    "Meeting tomorrow morning at 10 AM regarding project X",
    "Critical security vulnerability detected on staging server",
    "Question about the invoice for last month's services",
    "Your subscription is expiring soon - renew now!",
    "Team lunch invitation for Friday"
]

print("Incoming Emails and their Priorities:")
for i, email in enumerate(incoming_emails):
    priority = classify_email_priority(email)
    print(f"Email {i+1}: {email} -> Priority: {priority}")
```

OUTPUT:

```
Input- Customer Review: The production server is down. Customers are unable to place orders.
Output- Low Priority

Incoming Emails and their Priorities:
Email 1: URGENT: Production database offline, website not loading! -> Priority: High Priority
Email 2: Request for new feature in dashboard -> Priority: Low Priority
Email 3: Weekly newsletter - Latest updates and promotions -> Priority: Low Priority
Email 4: Server maintenance scheduled for next month -> Priority: Low Priority
Email 5: Customer reported bug in login process - intermittent issue -> Priority: Low Priority
Email 6: Meeting tomorrow morning at 10 AM regarding project X -> Priority: Low Priority
Email 7: Critical security vulnerability detected on staging server -> Priority: Low Priority
Email 8: Question about the invoice for last month's services -> Priority: Medium Priority
Email 9: Your subscription is expiring soon - renew now! -> Priority: Low Priority
Email 10: Team lunch invitation for Friday -> Priority: Low Priority
```

Task 2: Few-Shot Prompt

Generate a code for email priority classification that prioritizes the generated emails into labels: High Priority,Low Priority or Medium Priority. Examples1: Input-System outage affecting all customers. Output-High Priority; Example 2: Input-Please review the document and share feedback by evening. Output-Medium Priority; Example 3: Input-Seasonal greetings from the HR team. Output-Low Priority. Now analyze the example and generate 10 incoming emails and prioritize them into labels. Generate a efficient and readable code.

CODE:

```python
import re

# Function to classify email priority (re-using the refined function)
def classify_email_priority(email_subject_body):
    text = email_subject_body.lower()

    # High Priority Keywords
    high_priority_keywords = [
        "server down", "production outage", "critical issue", "urgent",
        "security breach", "data loss", "customers unable", "payment failed",
        "system error", "cannot access", "firewall alert", "offline",
        "website not loading", "database offline", "system unavailable",
        "critical vulnerability", "production server down", "unable to place orders",
        "critical security", "emergency", "downtime"
    ]
    for keyword in high_priority_keywords:
        if keyword in text:
            return "High Priority"

    # Medium Priority Keywords
    medium_priority_keywords = [
        "bug report", "feature request", "follow up", "meeting reminder",
        "update required", "performance issue", "access request", "question about",
        "scheduled maintenance", "intermittent issue", "invoice", "subscription expiring",
        "new feature", "maintenance scheduled", "meeting", "security vulnerability",
        "subscription is expiring", "proposal", "review request"
    ]
    for keyword in medium_priority_keywords:
        if keyword in text:
```

```python
            return "Low Priority"

# Generate a new set of 10 sample incoming emails
new_incoming_emails = [
    "Emergency: All systems offline due to power outage!",
    "Feature request: Add dark mode to the application",
    "Company picnic invitation for next month",
    "System performance degradation detected in staging environment",
    "URGENT: Database connection failure in production!",
    "Question about my recent order #12345",
    "Reminder: Team sync meeting tomorrow at 11 AM",
    "Your annual subscription is due for renewal",
    "Security alert: Minor vulnerability reported in old library",
    "Newsletter: Our latest blog posts and updates"
]

# Store results for summary
new_results_summary = {'High Priority': 0, 'Medium Priority': 0, 'Low Priority': 0}
new_classified_emails = []

print("Incoming Emails and their Priorities (New Set):")
for i, email in enumerate(new_incoming_emails):
    priority = classify_email_priority(email)
    new_classified_emails.append({'email': email, 'priority': priority})
    new_results_summary[priority] += 1
    print(f"Email {i+1}: {email} -> Priority: {priority}")

print("\nSummary of Email Prioritization (New Set):")
for priority_level, count in new_results_summary.items():
    print(f"{priority_level}: {count} emails")
```

OUTPUT:

```
Incoming Emails and their Priorities (New Set):
Email 1: Emergency: All systems offline due to power outage! -> Priority: High Priority
Email 2: Feature request: Add dark mode to the application -> Priority: Medium Priority
Email 3: Company picnic invitation for next month -> Priority: Low Priority
Email 4: System performance degradation detected in staging environment -> Priority: Low Priority
Email 5: URGENT: Database connection failure in production! -> Priority: High Priority
Email 6: Question about my recent order #12345 -> Priority: Medium Priority
Email 7: Reminder: Team sync meeting tomorrow at 11 AM -> Priority: Medium Priority
Email 8: Your annual subscription is due for renewal -> Priority: Low Priority
Email 9: Security alert: Minor vulnerability reported in old library -> Priority: Low Priority
Email 10: Newsletter: Our latest blog posts and updates -> Priority: Low Priority

Summary of Email Prioritization (New Set):
High Priority: 2 emails
Medium Priority: 3 emails
Low Priority: 5 emails
```

JUSTIFICATION:

One-shot classification uses only one example, so the rules are simpler and sometimes miss edge cases. Few-shot classification uses multiple examples, so it understands priorities more clearly. Few-shot produces more consistent labels across different types of emails. One-shot may misclassify borderline emails. Therefore, few-shot gives more accurate and reliable outputs than one-shot.

Question 3: Student Query Routing System

Task: One Shot Prompt

Generate a code for student Query Routing system. It should must route student queries to Admissions, Exams, Academics or Placements. Example: Input-What is the last date to apply for the MBA program? Output- Admissions; Now analyze the example and generate 10 random Student queries and route them with suitable labels. The code must be efficient and readable.

CODE:

```python
import re

# Function to classify student queries
def classify_student_query(query):
    text = query.lower()

    # Admissions Keywords
    admissions_keywords = [
        "apply", "application", "admission", "enroll", "eligibility",
        "requirements", "deadline", "fee", "mba program", "bsc program",
        "phd program", "how to get in", "start date", "course fees",
        "scholarship", "aid", "documents required"
    ]
    for keyword in admissions_keywords:
        if keyword in text:
            return "Admissions"

    # Exams Keywords
    exams_keywords = [
        "exam", "test", "quiz", "midterm", "final", "schedule",
        "date sheet", "results", "grade", "passing marks", "invigilator",
        "reschedule", "examination center", "hall ticket", "admit card"
    ]
    for keyword in exams_keywords:
        if keyword in text:
            return "Exams"

    # Academics Keywords
    academics_keywords = [
```

```python
    # Academics Keywords
    academics_keywords = [
        "course", "subject", "curriculum", "syllabus", "lecture",
        "assignment", "project", "faculty", "professor", "research",
        "program", "degree", "change major", "academic advisor",
        "course material", "gpa", "transcript", "class schedule"
    ]
    for keyword in academics_keywords:
        if keyword in text:
            return "Academics"

    # Placements Keywords (default if no other keywords match, or explicitly mentioned)
    placements_keywords = [
        "job", "internship", "placement", "career", "recruitment",
        "company", "interview", "cv", "resume", "mock interview",
        "industry", "job fair", "employment", "salary", "offer letter"
    ]
    for keyword in placements_keywords:
        if keyword in text:
            return "Placements"

    # Default to General Inquiry or similar if no specific keywords match
    return "General Inquiry"

# Generate 10 sample student queries
sample_student_queries = [
    "What is the last date to apply for the MBA program?",
    "When are the final exam schedules released for engineering courses?",
    "Can I get the syllabus for the 'Introduction to AI' course?",
```

```
# Generate 10 sample student queries
sample_student_queries = [
    "What is the last date to apply for the MBA program?",
    "When are the final exam schedules released for engineering courses?",
    "Can I get the syllabus for the 'Introduction to AI' course?",
    "What companies are participating in the upcoming placement drive?",
    "I need help with my assignment for advanced calculus.",
    "How can I check my semester exam results?",
    "Is there any scholarship available for international students?",
    "I want to change my major from Physics to Chemistry, what is the procedure?",
    "When is the career fair for the IT department?",
    "What documents are required for admission to the Master's program?"
]

# Store results for summary
query_results_summary = {'Admissions': 0, 'Exams': 0, 'Academics': 0, 'Placements': 0, 'General Inquiry': 0}
classified_queries = []

print("Student Queries and their Routing:")
for i, query in enumerate(sample_student_queries):
    department = classify_student_query(query)
    classified_queries.append({'query': query, 'department': department})
    query_results_summary[department] += 1
    print(f"Query {i+1}: {query} -> Department: {department}")

print("\nSummary of Query Routing:")
for department_name, count in query_results_summary.items():
    if count > 0: # Only print departments that have queries routed to them
        print(f"{department_name}: {count} queries")
```

OUTPUT:

```
Student Queries and their Routing:
Query 1: What is the last date to apply for the MBA program? -> Department: Admissions
Query 2: When are the final exam schedules released for engineering courses? -> Department: Exams
Query 3: Can I get the syllabus for the 'Introduction to AI' course? -> Department: Academics
Query 4: What companies are participating in the upcoming placement drive? -> Department: Placements
Query 5: I need help with my assignment for advanced calculus. -> Department: Academics
Query 6: How can I check my semester exam results? -> Department: Exams
Query 7: Is there any scholarship available for international students? -> Department: Admissions
Query 8: I want to change my major from Physics to Chemistry, what is the procedure? -> Department: General Inquiry
Query 9: When is the career fair for the IT department? -> Department: Placements
Query 10: What documents are required for admission to the Master's program? -> Department: Admissions

Summary of Query Routing:
Admissions: 3 queries
Exams: 2 queries
Academics: 2 queries
Placements: 2 queries
General Inquiry: 1 queries
```

JUSTIFICATION:

The output correctly routes each query to Admissions, Exams, Academics, or Placements based on keywords. Most queries are classified accurately, showing the logic is working as intended. Only one query goes to General Inquiry, which is reasonable because it does not clearly match a category. The distribution across departments looks balanced and realistic. Overall, the output demonstrates that the routing system is reliable and effective.

Question 4: Chatbot Question Type Detection

Task: Few Shot Prompt:

Generate a code for chatbot question type detection. It must identify whether a user query is Informational, Transactional, Complaint or Feedback. Example1: Input-What are your customer support working hours? Output- Informational; Example 2: Input-Please cancel my subscription immediately. Output- Transactional; Example 3: Input-The app keeps crashing after the latest update. Ouput-Complaint ; Example 4: Input-I really like the new dark mode feature. Output-Feedback Now analyze the examples and generate 10 random queries and identify them as one of the labels. Generate a efficient and readable code.

CODE:

```python
import re

# Function to classify chatbot question types
def classify_query_type(query):
    text = query.lower()

    # Informational Keywords
    informational_keywords = [
        "what", "how", "when", "where", "tell me about", "information about",
        "explain", "define", "hours", "address", "contact", "details",
        "faq", "guide", "manual", "working hours", "policy", "status"
    ]
    for keyword in informational_keywords:
        if keyword in text:
            return "Informational"

    # Transactional Keywords
    transactional_keywords = [
        "buy", "order", "purchase", "book", "reserve", "sign up",
        "subscribe", "cancel", "change", "update", "add to cart",
        "checkout", "pay", "process", "request", "apply for", "create account",
        "cancel my subscription", "renew", "upgrade"
    ]
    for keyword in transactional_keywords:
        if keyword in text:
            return "Transactional"

    # Complaint Keywords
    complaint_keywords = [
```

```python
    # Complaint Keywords
    complaint_keywords = [
        "problem", "issue", "error", "not working", "broken",
        "dissatisfied", "bad service", "unhappy", "failed", "bug",
        "wrong", "stuck", "can't", "slow", "crashing", "keeps crashing",
        "outage", "malfunction", "glitch"
    ]
    for keyword in complaint_keywords:
        if keyword in text:
            return "Complaint"

    # Feedback Keywords
    feedback_keywords = [
        "feedback", "suggestion", "idea", "improve", "like", "dislike",
        "review", "comment", "experience", "think about", "opinion",
        "really like", "love", "great feature", "hate", "terrible"
    ]
    for keyword in feedback_keywords:
        if keyword in text:
            return "Feedback"

    # Default to Informational if no other keywords match
    return "Informational"

# Generate 10 new sample user queries based on the examples
sample_user_queries = [
    "What are your customer support working hours?", # Informational
    "Please cancel my subscription immediately.", # Transactional
    "The app keeps crashing after the latest update.", # Complaint
```

```python
    "I really like the new dark mode feature.", # Feedback
    "How can I track the status of my order?", # Informational
    "I want to purchase a premium plan.", # Transactional
    "My payment keeps failing, and I can't complete the checkout.", # Complaint
    "Your new website design is very confusing.", # Feedback
    "Can I upgrade my account to the professional tier?", # Transactional
    "This bug is making the application unusable.", # Complaint
    "Tell me about your privacy policy.", # Informational
    "I have a suggestion for a new feature.", # Feedback
    "I need to return an item, what's the process?", # Transactional
    "The customer service I received was unacceptable.", # Complaint
    "I think your product is excellent, keep up the good work!" # Feedback
]

# Store results for summary
query_type_summary = {'Informational': 0, 'Transactional': 0, 'Complaint': 0, 'Feedback': 0}
classified_user_queries = []

print("User Queries and their Detected Types:")
for i, query in enumerate(sample_user_queries):
    query_type = classify_query_type(query)
    classified_user_queries.append({'query': query, 'type': query_type})
    query_type_summary[query_type] += 1
    print(f"Query {i+1}: {query} -> Type: {query_type}")

print("\nSummary of Query Type Detection:")
for q_type, count in query_type_summary.items():
    if count > 0: # Only print types that have queries routed to them
        print(f"{q_type}: {count} queries")
```

OUTPUT:

```
...  User Queries and their Detected Types:
     Query 1: What are your customer support working hours? -> Type: Informational
     Query 2: Please cancel my subscription immediately. -> Type: Transactional
     Query 3: The app keeps crashing after the latest update. -> Type: Transactional
     Query 4: I really like the new dark mode feature. -> Type: Feedback
     Query 5: How can I track the status of my order? -> Type: Informational
     Query 6: I want to purchase a premium plan. -> Type: Transactional
     Query 7: My payment keeps failing, and I can't complete the checkout. -> Type: Transactional
     Query 8: Your new website design is very confusing. -> Type: Informational
     Query 9: Can I upgrade my account to the professional tier? -> Type: Transactional
     Query 10: This bug is making the application unusable. -> Type: Complaint
     Query 11: Tell me about your privacy policy. -> Type: Informational
     Query 12: I have a suggestion for a new feature. -> Type: Feedback
     Query 13: I need to return an item, what's the process? -> Type: Informational
     Query 14: The customer service I received was unacceptable. -> Type: Informational
     Query 15: I think your product is excellent, keep up the good work! -> Type: Informational

     Summary of Query Type Detection:
     Informational: 7 queries
     Transactional: 5 queries
     Complaint: 1 queries
     Feedback: 2 queries
```

JUSTIFICATION:

The prompt gives clear few-shot examples, so the model learns how each label should look. The output correctly matches informational queries with Informational. Action-based requests are consistently classified as Transactional. Problem-related sentences are labeled as Complaint, and opinion-based sentences as Feedback. Overall, the output aligns with the examples, showing the few-shot guidance is effective and consistent.

Question 5: Emotion Detection in Text

Task 1: One Shot Prompt:

Generate a code for Emotion Detection in Text. It needs to detect emotions : Happy, Sad,Angry,Anxious ,Neutral. Example: Input-I finally got the job I worked so hard for! , Output-Happy Now analyze the example and generate 10 random emotion samples texts and identify the emotion. Generate a efficient and readable code.

CODE:

```python
import re

# Function to detect emotions in text
def detect_emotion(text):
    normalized_text = text.lower()

    # Happy Keywords
    happy_keywords = [
        "happy", "excited", "joy", "great", "amazing", "wonderful", "fantastic",
        "love it", "good news", "finally", "awesome", "thrilled", "pleased",
        "excellent", "perfect", "yay", "hooray", "success", "accomplished"
    ]
    for keyword in happy_keywords:
        if keyword in normalized_text:
            return "Happy"

    # Sad Keywords
    sad_keywords = [
        "sad", "unhappy", "depressed", "heartbroken", "terrible", "miserable",
        "awful", "unfortunately", "crying", "disappointed", "lost", "regret",
        "bad news", ":'(", "upset", "gloomy", "sorrow"
    ]
    for keyword in sad_keywords:
        if keyword in normalized_text:
            return "Sad"

    # Angry Keywords
    angry_keywords = [
        "angry", "furious", "mad", "frustrated", "hate", "terrible service",
```

```python
        "angry", "furious", "mad", "frustrated", "hate", "terrible service",
        "unacceptable", "problem", "issue", "broken", "crashing", "inefficient",
        "rage", ">:(", "annoyed", "irritated", "disgusted"
    ]
    for keyword in angry_keywords:
        if keyword in normalized_text:
            return "Angry"

    # Anxious Keywords
    anxious_keywords = [
        "anxious", "worried", "nervous", "stress", "stressful", "fear",
        "scared", "concerned", "uneasy", "apprehensive", "dreadful",
        "tension", "pressure", "panic", "overwhelmed", "jittery", "doubt"
    ]
    for keyword in anxious_keywords:
        if keyword in normalized_text:
            return "Anxious"

    # Neutral (default if no other keywords match)
    return "Neutral"

# Generate 10 random emotion samples texts
sample_texts = [
    "I finally got the job I worked so hard for!", # Happy
    "My pet isn't feeling well, and I'm very worried.", # Anxious
    "The train was delayed for two hours, I'm so frustrated.", # Angry
    "The weather forecast for tomorrow is partly cloudy with a chance of rain.", # Neutral
    "I just won the lottery! This is absolutely amazing!", # Happy
    "I feel so down after hearing the bad news, I can't stop crying.", # Sad
```

```
        "The weather forecast for tomorrow is partly cloudy with a chance of rain.", # Neutral
        "I just won the lottery! This is absolutely amazing!", # Happy
        "I feel so down after hearing the bad news, I can't stop crying.", # Sad
        "This software update broke everything! I'm absolutely furious.", # Angry
        "I have a big presentation tomorrow, and I'm feeling a lot of pressure.", # Anxious
        "The meeting went smoothly, and we covered all the agenda items.", # Neutral
        "I'm so disappointed with the outcome, it's just not what I expected.", # Sad
        "The new feature is perfect, I love it!", # Happy
        "I'm concerned about the security breach mentioned in the news.", # Anxious
        "The internet connection is constantly dropping, it's a huge problem!", # Angry
        "My routine consists of waking up, working, and then relaxing.", # Neutral
        "I'm extremely thrilled to announce our new product launch!" # Happy
]

# Store results for summary
emotion_summary = {'Happy': 0, 'Sad': 0, 'Angry': 0, 'Anxious': 0, 'Neutral': 0}
classified_emotions = []

print("Texts and their Detected Emotions:")
for i, text in enumerate(sample_texts):
    emotion = detect_emotion(text)
    classified_emotions.append({'text': text, 'emotion': emotion})
    emotion_summary[emotion] += 1
    print(f"Text {i+1}: {text} -> Emotion: {emotion}")

print("\nSummary of Emotion Detection:")
for emotion_type, count in emotion_summary.items():
    if count > 0: # Only print emotions that have been detected
        print(f"{emotion_type}: {count} texts")
```

OUTPUT:

```
Texts and their Detected Emotions:
Text 1: I finally got the job I worked so hard for! -> Emotion: Happy
Text 2: My pet isn't feeling well, and I'm very worried. -> Emotion: Anxious
Text 3: The train was delayed for two hours, I'm so frustrated. -> Emotion: Angry
Text 4: The weather forecast for tomorrow is partly cloudy with a chance of rain. -> Emotion: Neutral
Text 5: I just won the lottery! This is absolutely amazing! -> Emotion: Happy
Text 6: I feel so down after hearing the bad news, I can't stop crying. -> Emotion: Sad
Text 7: This software update broke everything! I'm absolutely furious. -> Emotion: Angry
Text 8: I have a big presentation tomorrow, and I'm feeling a lot of pressure. -> Emotion: Anxious
Text 9: The meeting went smoothly, and we covered all the agenda items. -> Emotion: Neutral
Text 10: I'm so disappointed with the outcome, it's just not what I expected. -> Emotion: Sad
Text 11: The new feature is perfect, I love it! -> Emotion: Happy
Text 12: I'm concerned about the security breach mentioned in the news. -> Emotion: Anxious
Text 13: The internet connection is constantly dropping, it's a huge problem! -> Emotion: Angry
Text 14: My routine consists of waking up, working, and then relaxing. -> Emotion: Neutral
Text 15: I'm extremely thrilled to announce our new product launch! -> Emotion: Happy

Summary of Emotion Detection:
Happy: 4 texts
Sad: 2 texts
Angry: 3 texts
Anxious: 3 texts
Neutral: 3 texts
```

Task 2: Few Shot Prompt:

Generate a code for Emotion Detection in Text. It needs to detect emotions : Happy, Sad,Angry,Anxious ,Neutral. Example 1: Input-I finally got the job I worked so hard for! , Output-Happy ; Example 2: Input-I feel completely alone and nothing seems to help. Output-Sad; Example 3: Input-This service is terrible and I am really frustrated. Output-Angry ; Example 4: Input-I keep worrying about my future and can't sleep. Output- Anxious ; Now analyze the examples and generate 10 random emotion samples texts and identify the emotion. Generate a efficient and readable code.

CODE:

```python
import re

# Function to detect emotions in text
def detect_emotion(text):
    normalized_text = text.lower()

    # Happy Keywords
    happy_keywords = [
        "happy", "excited", "joy", "great", "amazing", "wonderful", "fantastic",
        "love it", "good news", "finally", "awesome", "thrilled", "pleased",
        "excellent", "perfect", "yay", "hooray", "success", "accomplished"
    ]
    for keyword in happy_keywords:
        if keyword in normalized_text:
            return "Happy"

    # Sad Keywords
    sad_keywords = [
        "sad", "unhappy", "depressed", "heartbroken", "terrible", "miserable",
        "awful", "unfortunately", "crying", "disappointed", "lost", "regret",
        "bad news", ":'(", "upset", "gloomy", "sorrow", "alone"
    ]
    for keyword in sad_keywords:
        if keyword in normalized_text:
            return "Sad"

    # Angry Keywords
    angry_keywords = [
        "angry", "furious", "mad", "frustrated", "hate", "terrible service",
```

```python
    angry_keywords = [
        "angry", "furious", "mad", "frustrated", "hate", "terrible service",
        "unacceptable", "problem", "issue", "broken", "crashing", "inefficient",
        "rage", ">:(", "annoyed", "irritated", "disgusted"
    ]
    for keyword in angry_keywords:
        if keyword in normalized_text:
            return "Angry"

    # Anxious Keywords
    anxious_keywords = [
        "anxious", "worried", "nervous", "stress", "stressful", "fear",
        "scared", "concerned", "uneasy", "apprehensive", "dreadful",
        "tension", "pressure", "panic", "overwhelmed", "jittery", "doubt", "worrying"
    ]
    for keyword in anxious_keywords:
        if keyword in normalized_text:
            return "Anxious"

    # Neutral (default if no other keywords match)
    return "Neutral"

# Generate 10 random emotion samples texts based on user examples
sample_texts = [
    "I finally got the job I worked so hard for!", # Happy
    "I feel completely alone and nothing seems to help.", # Sad
    "This service is terrible and I am really frustrated.", # Angry
    "I keep worrying about my future and can't sleep.", # Anxious
    "The sun is shining today, what a wonderful day!", # Happy
```

```python
# Generate 10 random emotion samples texts based on user examples
sample_texts = [
    "I finally got the job I worked so hard for!", # Happy
    "I feel completely alone and nothing seems to help.", # Sad
    "This service is terrible and I am really frustrated.", # Angry
    "I keep worrying about my future and can't sleep.", # Anxious
    "The sun is shining today, what a wonderful day!", # Happy
    "My favorite coffee shop closed down, I'm quite sad about it.", # Sad
    "The internet connection is constantly dropping, it's a huge problem!", # Angry
    "I have a major exam tomorrow and I'm very nervous.", # Anxious
    "The meeting is scheduled for 3 PM on Tuesday.", # Neutral
    "I'm absolutely thrilled with the new software update!" # Happy
]

# Store results for summary
emotion_summary = {'Happy': 0, 'Sad': 0, 'Angry': 0, 'Anxious': 0, 'Neutral': 0}
classified_emotions = []

print("Texts and their Detected Emotions:")
for i, text in enumerate(sample_texts):
    emotion = detect_emotion(text)
    classified_emotions.append({'text': text, 'emotion': emotion})
    emotion_summary[emotion] += 1
    print(f"Text {i+1}: {text} -> Emotion: {emotion}")

print("\nSummary of Emotion Detection:")
for emotion_type, count in emotion_summary.items():
    if count > 0: # Only print emotions that have been detected
        print(f"{emotion_type}: {count} texts")
```

OUTPUT:

```
••• Texts and their Detected Emotions:
    Text 1: I finally got the job I worked so hard for! -> Emotion: Happy
    Text 2: I feel completely alone and nothing seems to help. -> Emotion: Sad
    Text 3: This service is terrible and I am really frustrated. -> Emotion: Sad
    Text 4: I keep worrying about my future and can't sleep. -> Emotion: Anxious
    Text 5: The sun is shining today, what a wonderful day! -> Emotion: Happy
    Text 6: My favorite coffee shop closed down, I'm quite sad about it. -> Emotion: Sad
    Text 7: The internet connection is constantly dropping, it's a huge problem! -> Emotion: Angry
    Text 8: I have a major exam tomorrow and I'm very nervous. -> Emotion: Anxious
    Text 9: The meeting is scheduled for 3 PM on Tuesday. -> Emotion: Neutral
    Text 10: I'm absolutely thrilled with the new software update! -> Emotion: Happy

    Summary of Emotion Detection:
    Happy: 3 texts
    Sad: 3 texts
    Angry: 1 texts
    Anxious: 2 texts
    Neutral: 1 texts
```

JUSTIFICATION:

One-shot output follows only one example, so emotion detection works but is more dependent on simple keyword matches. Few-shot output is more accurate because multiple examples clarify the differences between Happy, Sad, Angry, and Anxious. Few-shot results show better handling of similar emotions like Sad vs Anxious. One-shot may misclassify some borderline texts due to limited guidance. Overall, few-shot produces more consistent and reliable emotion classification than one-shot.