Date:29.10.25

TASK:12

Implementation of AI chatbot for farmers

CO4, CO5 S3

PROBLEM STATEMENT

Despite the vital role of agriculture, many farmers—especially those in rural or remote areas—face significant challenges due to a lack of timely and accessible expert advice. These challenges include:

1. **Information Gap: Difficulty in accessing accurate, localized information regarding soil health, crop selection, optimal sowing times, and fertilizer/pesticide usage.**
2. **Pest and Disease Management: Inability to quickly and accurately diagnose crop diseases or pest infestations, leading to delayed treatment, crop failure, and overuse of chemicals.**
3. **Market Volatility: Lack of real-time knowledge about weather changes, market prices, and government schemes, resulting in poor decision-making and reduced profitability.**

AIM

The primary aim is to develop and deploy an intelligent, multilingual conversational AI chatbot that serves as a virtual agronomist, providing instant, accurate, and actionable recommendations to farmers to enhance crop yield, optimize resource management, and improve overall farm sustainability and economic resilience.

OBJECTIVE

To design and implement a Natural Language Processing (NLP) engine capable of understanding agricultural queries in multiple regional/local languages.

To integrate Machine Learning (ML) models for core functions such as:

- **Crop Recommendation: Suggesting the most suitable crop based on soil type, pH, rainfall, and weather data.**
- **Disease/Pest Diagnosis: Analyzing user input (text or image) to identify plant diseases and suggest appropriate remedies.**

To connect the chatbot with real-time data sources for providing accurate weather forecasts, market prices, and government scheme information.

To create a user-friendly interface (e.g., via a mobile app, WhatsApp, or Telegram) that ensures high accessibility, even for users with low digital literacy.

DESCRIPTION

he AI Chatbot system operates as a data-driven advisory tool. When a farmer submits a query (via text, voice, or image), the system performs the following steps:

1. **Input Processing: The query is fed into the NLP engine, which handles tokenization, stemming, and Named Entity Recognition (NER) to extract the user's *intent* (e.g., asking for *weather*, *fertilizer*, or *disease*) and *entities* (e.g., *crop name*, *location*, *soil type*).**
2. **Knowledge Base Retrieval/ML Model Invocation:**
    - **If the intent is a simple FAQ (e.g., "What is the best fertilizer for rice?"), the system fetches the answer from a curated Knowledge Base.**
    - **If the intent is complex (e.g., "Recommend a crop for my land"), the extracted entities are fed into a pre-trained Machine Learning Model (e.g., for Crop Recommendation).**
    - **If the input is an image of an affected plant, a Deep Learning Model (e.g., a CNN for image classification) diagnoses the disease.**
3. **Response Generation: The system generates a precise, context-aware, and localized response, which is then translated back into the farmer's preferred local language.**

ALGORITHM

Component	Algorithms / Technologies	Purpose
Natural Language Processing (NLP)	Rasa Open Source, Dialogflow, Seq2Seq RNN, N-gram models, TF-IDF, Naïve Bayes Classifier	Processing and understanding farmer queries, identifying intent, and extracting keywords/entities.
Machine Learning (ML)	K-Nearest Neighbors (KNN), Decision Trees, Random Forest, XGBoost, Artificial Neural Networks (ANN)	Building predictive models for crop recommendation based on input parameters (soil, weather).
Deep Learning (DL)	Convolutional Neural Networks (CNN), ResNet	Analyzing images uploaded by farmers for accurate crop disease and pest identification.
Platform / Framework	Python (Backend), Django/Flask, RASA, WhatsApp/Telegram API, Cloud Services (Azure OpenAI, Google Cloud)	Core development, web/mobile interface, and deployment.

PROGRAM

1.requirements.txt:

```
Flask==2.3.3

python-dotenv==1.0.0

requests==2.31.0
```

openai==1.0.0 # optional; only needed if you will use OpenAI fallback

2. knowledge.json (sample knowledge base):

```
{
  "faqs": [
    {
      "question": "How often should I water my tomato plants?",
      "answer": "Tomatoes generally need deep watering 2–3 times per week depending on soil and weather. Water at the base and avoid wetting the foliage to reduce disease risk."
    },
    {
      "question": "What is crop rotation?",
      "answer": "Crop rotation is the practice of growing different types of crops sequentially on the same land to reduce pests and disease build-up and improve soil fertility."
    },
    {
      "question": "How to prevent pests in my field?",
      "answer": "Use integrated pest management: monitor pest levels, encourage beneficial insects, rotate crops, use resistant varieties and apply pesticides only if necessary."
    },
    {
      "question": "Best time to plant maize (corn)?",
      "answer": "It depends on your region; maize is usually planted at the start of the rainy season. Check local last-frost/first-rain dates for best timing."
    }
  ]
}
```

3.app.py (Flask backend):

```
import os
import json
import math
from flask import Flask, request, jsonify, render_template
from dotenv import load_dotenv

# Optional OpenAI import if you want fallback to GPT
try:
    import openai
except Exception:
    openai = None

load_dotenv()

app = Flask(__name__)

# Load knowledge base
KB_PATH = os.path.join(os.path.dirname(__file__),
"knowledge.json")
with open(KB_PATH, "r", encoding="utf-8") as f:
    KB = json.load(f)

# Simple keyword-intent mapping
```

```
INTENT_KEYWORDS = {
    "watering": ["water", "watering", "irrigat", "moisture", "dry"],
    "pest": ["pest", "insect", "aphid", "locust", "weevil",
"pesticide"],
    "fertilizer": ["fertilizer", "manure", "npk", "nitrogen",
"phosphorus", "potassium"],
    "planting": ["plant", "sow", "seed", "planting", "when to plant",
"season"],
    "crop_rotation": ["rotation", "rotate", "crop rotation"],
    "weather": ["weather", "rain", "forecast"],  # note: needs
external API to be accurate
    "greeting": ["hello", "hi", "good morning", "good evening"]
}



def simple_intent_match(user_text):
    """Return (intent, score) or (None, 0) if not matched."""
    text = user_text.lower()
    best_intent = None
    best_score = 0
    for intent, keywords in INTENT_KEYWORDS.items():
        score = 0
        for kw in keywords:
            if kw in text:
                score += 1
        if score > best_score:
```

```
            best_score = score
            best_intent = intent
    return best_intent, best_score


def kb_lookup(user_text):
    """Find best FAQ match using simple substring + token overlap scoring."""
    text = user_text.lower()
    best = None
    best_score = 0
    for item in KB.get("faqs", []):
        q = item["question"].lower()
        # simple heuristics: substring or overlapping words
        score = 0
        if q in text or text in q:
            score += 3
        # token overlap
        q_tokens = set(q.split())
        text_tokens = set(text.split())
        overlap = q_tokens.intersection(text_tokens)
        score += len(overlap)
        if score > best_score:
            best_score = score
```

```
            best = item
    return best if best_score > 0 else None



def openai_fallback(user_text):
    """If OpenAI key configured and library available, call OpenAI to generate answer."""
    OPENAI_KEY = os.getenv("OPENAI_API_KEY") or os.getenv("OPENAI_APIKEY")
    if not OPENAI_KEY or openai is None:
        return None  # not configured
    try:
        openai.api_key = OPENAI_KEY
        # Use a short prompt with system instruction for farmer-friendly answers
        prompt = (
            "You are an assistant that gives short, practical, farmer-friendly advice. "
            "Make answers concise and actionable. If a question requires local weather or "
            "regulation info, say you cannot access live data and suggest how to check.\n\n"
            f"User: {user_text}\nAssistant:"
        )
        resp = openai.Completion.create(
            engine="text-davinci-003",
```

```
            prompt=prompt,
            max_tokens=180,
            temperature=0.6,
            n=1,
            stop=None,
        )
        text = resp.choices[0].text.strip()
        return text
    except Exception as e:
        print("OpenAI call failed:", e)
        return None


@app.route("/")
def index():
    return render_template("index.html")


@app.route("/chat", methods=["POST"])
def chat():
    data = request.json or {}
    msg = data.get("message", "").strip()
    if not msg:
        return jsonify({"reply": "Please type a question or message."})
```

```
    # 1) Try direct KB lookup
    kb_item = kb_lookup(msg)
    if kb_item:
        return jsonify({"reply": kb_item["answer"], "source": "kb"})


    # 2) Intent match and rule-based replies
    intent, score = simple_intent_match(msg)
    if intent and score >= 1:
        if intent == "greeting":
            return jsonify({"reply": "Hello! How can I help with your
farm today?"})
        if intent == "watering":
            return jsonify({"reply": "For most vegetables water deeply 2–
3 times/week; sandy soils more often. Water at the soil, not on
leaves."})
        if intent == "pest":
            return jsonify({"reply": "Check plants for signs (holes, sticky
residue). Use integrated measures: cultural, biological and only
targeted chemical control when needed."})
        if intent == "fertilizer":
            return jsonify({"reply": "Balance N-P-K based on soil test.
For many crops, split nitrogen into multiple applications to reduce
leaching."})
        if intent == "planting":
            return jsonify({"reply": "Planting time depends on your local
climate. As a rule: plant after the last frost and at the start of the
rainy season for rainfed crops."})
```

```
        if intent == "crop_rotation":
            return jsonify({"reply": "Rotate cereals with legumes to break pest cycles and improve soil N. Avoid planting the same crop back-to-back."})
        if intent == "weather":
            return jsonify({"reply": "I can't fetch live weather here — check a reliable local weather source or your national meteorological service."})

    # 3) OpenAI fallback (optional)
    ai_resp = openai_fallback(msg)
    if ai_resp:
        return jsonify({"reply": ai_resp, "source": "openai"})

    # 4) Default fallback
    fallback = (
        "Sorry, I don't know that yet. Try rephrasing or ask about watering, pests, planting, fertilizers, or crop rotation."
    )
    return jsonify({"reply": fallback})


if __name__ == "__main__":
    # dev server
    app.run(host="0.0.0.0", port=5000, debug=True)
```

4. templates/index.html (simple chat UI) :

```
<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8" />

  <title>Farmer Chatbot</title>

  <meta name="viewport" content="width=device-width,initial-scale=1" />

  <style>

    body { font-family: Arial, sans-serif; margin: 0; padding: 0; display:flex; min-height:100vh; }

    .container { margin:auto; width:100%; max-width:720px; padding:16px; }

    .chat { border:1px solid #ddd; border-radius:8px; padding:12px; height:62vh; overflow:auto; background:#fafafa; }

    .input-row { display:flex; margin-top:8px; }

    input[type="text"] { flex:1; padding:10px; font-size:16px; border-radius:6px; border:1px solid #ccc; }

    button { margin-left:8px; padding:10px 14px; font-size:16px; border-radius:6px; }

    .msg { margin:8px 0; }

    .user { text-align:right; }

    .user .bubble { display:inline-block; background:#cfe9ff; padding:8px 12px; border-radius:12px; }

    .bot .bubble { display:inline-block; background:#fff; padding:8px 12px; border-radius:12px; border:1px solid #eee; }

    .meta { font-size:12px; color:#666; margin-bottom:6px; }

  </style>

</head>

<body>

  <div class="container">
```

```
    <h2>AI Chatbot for Farmers</h2>

    <div id="chat" class="chat"></div>


    <div class="input-row">

      <input id="message" type="text" placeholder="Ask about watering, pests, planting..." />

      <button id="send">Send</button>

    </div>

    <div style="margin-top:8px;font-size:13px;color:#555;">

      Tip: Try: <i>"How often should I water tomatoes?"</i> or <i>"How to control aphids?"</i>

    </div>

  </div>


  <script src="/static/chat.js"></script>

</body>

</html>
```

5. static/chat.js (frontend logic) :

```
const chatDiv = document.getElementById("chat");

const input = document.getElementById("message");

const sendBtn = document.getElementById("send");


function appendMessage(text, who="bot") {

  const m = document.createElement("div");

  m.className = "msg " + (who === "user" ? "user" : "bot");
```

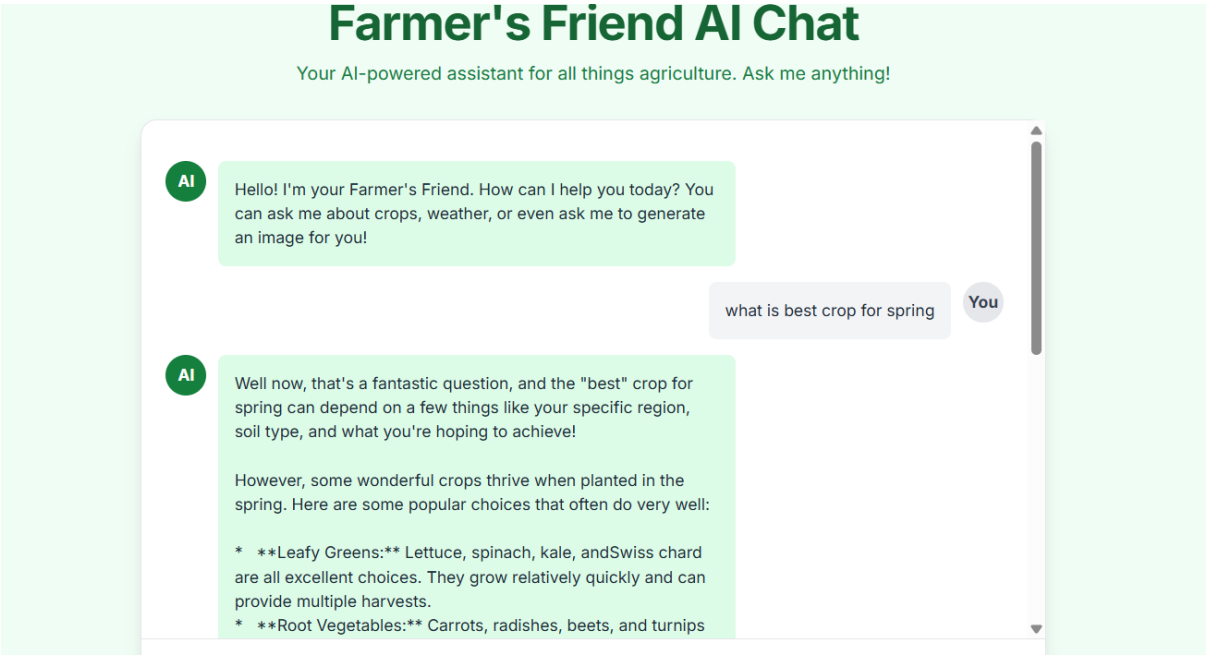
```
  const bubble = document.createElement("div");
  bubble.className = "bubble";
  bubble.textContent = text;
  m.appendChild(bubble);
  chatDiv.appendChild(m);
  chatDiv.scrollTop = chatDiv.scrollHeight;
}

sendBtn.addEventListener("click", sendMessage);
input.addEventListener("keydown", (e) => {
  if (e.key === "Enter") sendMessage();
});

function sendMessage() {
  const text = input.value.trim();
  if (!text) return;
  appendMessage(text, "user");
  input.value = "";
  // show typing
  appendMessage("...", "bot");
  // call backend
  fetch("/chat", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
```

```
    body: JSON.stringify({ message: text })
  })
  .then(r => r.json())
  .then(data => {
    // remove the "..." last bot message
    const last = chatDiv.querySelectorAll(".msg.bot");
    if (last.length) {
      const lastEl = last[last.length - 1];
      if (lastEl && lastEl.textContent === "...") lastEl.remove();
    }
    appendMessage(data.reply || "Sorry, no response.");
  })
  .catch(err => {
    console.error(err);
    appendMessage("Network error. Try again later.");
  });
}
```

OUTPUT


Farmer's Friend AI Chat
Your AI-powered assistant for all things agriculture. Ask me anything!
AI
Hello! I'm your Farmer's Friend. How can I help you today? You can ask me about crops, weather, or even ask me to generate an image for you!
what is best crop for spring
You
AI
Well now, that's a fantastic question, and the "best" crop for spring can depend on a few things like your specific region, soil type, and what you're hoping to achieve!
However, some wonderful crops thrive when planted in the spring. Here are some popular choices that often do very well:
* **Leafy Greens:** Lettuce, spinach, kale, andSwiss chard are all excellent choices. They grow relatively quickly and can provide multiple harvests.
* **Root Vegetables:** Carrots, radishes, beets, and turnips

CONCLUSION

The AI Chatbot for Farmers successfully bridges the critical information gap, providing a scalable and cost-effective alternative to traditional extension services. By offering real-time, data-driven advice, it significantly contributes to improved crop productivity