Building Your ITS: From Concept to Code

A PRACTICAL GUIDE TO IMPLEMENTING YOUR FIRST INTELLIGENT TUTORING SYSTEM

CIS*3750 – SYSTEM ANALYSIS AND DESIGN IN APPLICATIONS

PRESENTED BY: RICARDO JARRIN

TUESDAY, OCTOBER 21, 2025

Agenda

- Part 1: The Foundation
  - Going over the basics of a learning system
  - Simulate system using rules-based flash card demo + code
- Part 2: The ML Menu
  - Going through the ML aspects presented in the Sept 16 Lecture
  - Simple programs meant to give you an idea of how to implement
- Part 3: Live Demo
  - Going through an ITS that uses knowledge tracing + code
- Notable Resources
- Q&A

Part 1: The Foundation

A RULES-BASED ITS

The Goal for Today

- You understand the **what** – the four main ITS components. Today, we focus on the **how**
- We'll build a complete, rules-based tutor from scratch, step-by-step
- Then, we'll show how to "level up" **any** of the four components with specific Machine Learning tools and code examples

The Simplest Analogy: A ‘Smart’ Flashcard Deck

- A regular deck is **static**. A “smart” deck (our ITS) pays attention and adapts
- It knows the topics (**Domain**), tracks your progress (**Student**), decides what to show you next (**Tutoring**), and handles the interaction (**UI**)

Our Case Study: A Simple Algebra Tutor

- **Domain**: Solving basic equations
- **Skills**: addition, multiplication, solving linear-equations
- **Dependency**: addition → multiplication → solving linear-equations
- We will now build this tutor using a simple, non-ML approach

Walkthrough Step 1: The Domain and Student Models (The Data)

- The Rulebook (Domain): We'll use simple JSON files to store out skills, questions, and dependencies.
- The Scorecard (Student): We'll use a separate JSON file for each student to track their mastery score

```
{
  "addition": [{"q": "5 + 8", "a": "13"}],
  "multiplication": [{"q": "3 * 7", "a": "21"}],
  "solving-linear-equations": [{"q": "2x + 3 = 7", "a": "2"}]
}
```

```
{
  "multiplication": ["addition"],
  "solving-linear-equations": ["multiplication"]
}
```

```
{
  "student_id": "alex",
  "mastery": {
    "addition": 0.0,
    "multiplication": 0.0,
    "solving-linear-equations": 0.0
  }
}
```

Walkthrough Step 2: The Tutoring & UI Models (The Logic)

- The Game Master (Tutoring): The core of our system will be a Python function with if/else logic
- The Game Console (UI): We'll start a Command-Line interface (CLI) using Python's print() and input() functions.

Walkthrough Step 3: The Code - Tying It All Together

The Game Master → tutor.py

tutor.py

```python
MASTERY_THRESHOLD = 0.8 # Define what "mastered" means

def choose_next_skill(student_model, domain, dependencies):
    """Finds the next skill to teach."""
    mastery = student_model.get('mastery', {})

    # Iterate through all available skills
    for skill in domain.keys():
        # 1. Check if the skill is already mastered
        if mastery.get(skill, 0.0) >= MASTERY_THRESHOLD:
            continue

        # 2. Check if prerequisites are met
        prereqs = dependencies.get(skill, [])
        prereqs_met = all(mastery.get(p, 0.0) >= MASTERY_THRESHOLD for p in
        prereqs)

        # 3. If not mastered AND prereqs are met, this is our skill!
        if prereqs_met:
            return skill

    return None # No unmastered skills with met prerequisites
```

The Main Loop → main.py

main.py

```python
def run_tutor(student_id):
    """The main engine of the ITS."""
    # 1. Load all data models from the 'data/' folder
    domain, deps = load_domain('data/')
    student = load_student_model('data/', student_id)

    # 2. Start the main tutoring loop
    while True:
        # 3. TUTOR MODEL makes a decision
        next_skill = choose_next_skill(student, domain, deps)

        if next_skill is None:
            print("Congratulations! You've mastered everything!")
            break

        # 4. UI MODEL presents a problem
        problem = random.choice(domain[next_skill])
        answer = input(f"Question on {next_skill}: {problem['q']} -> ")

        # 5. Update STUDENT MODEL and save progress
        is_correct = (answer.strip() == problem['a'])
        student = update_mastery(student, next_skill, is_correct)
        save_student_model('data/', student)
```

Part 2: The “ML Menu”

UPGRADING YOUR TUTOR

Level Up: Where to Add Machine Learning

- Our rules-based tutor works, but ML can make it smarter, more automated, and more adaptive
- Now, we'll spotlight one ML upgrade for each of the four components, complete with runnable code examples

ML for the Domain Model: Skill Discovery with *scikit-learn*

- **Problem**: Manually tagging hundreds of questions with the correct skill is tedious and doesn't scale well.
- **ML Solution**: Use unsupervised clustering to automatically group similar questions, helping to discover the underlying skills in your content.
- **Tool**: scikit-learn, a fundamental Python library for machine learning.

ML for the Domain Model: Difficulty Estimation

- **Problem**: We don't know if a new question is too easy or too hard.
- **ML Solution**: Train a regression model to predict a question's difficulty based on its features (e.g., length, word complexity).
- **Tool**: scikit-learn.

ML for the Student Model: Knowledge Tracing with *pyBKT*

- **Problem**: Our simple +/- score for student mastery isn't very nuanced. It doesn't account for lucky guesses or accidental slips
- **ML Solution**: Use Bayesian Knowledge Tracing (BKT) to model student knowledge as a probability, providing a more realistic estimate of their mastery. This is a form of knowledge tracing
- **Tool**: pyBKT, a Python library dedicated to BKT modeling

ML for the Student Model: Recommendation

- **Problem**: Our tutor's rule for "what's next" is basic. It doesn't know what *similar* students found helpful.
- **ML Solution**: Use collaborative filtering to recommend the next skill based on the success patterns of other students.
- **Tool**: scikit-surprise

ML for the Tutoring Model: Feedback Personalization

- **Problem**: Our rules-based tutor's feedback is generic. It can't adapt its teaching style to what works best for an individual student
- **ML Solution**: Use a classification model to predict which type of feedback (e.g., a text hint vs. a worked example) is most likely to help a student, given their current state. This is a form of feedback personalization
- **Tool**: scikit-learn, for training a simple and effective classification model

ML for the Tutoring Model: Policy Learning

- **Problem**: Our "if/else" logic is just a simple strategy. What is the *optimal* teaching policy to maximize long-term learning?
- **ML Solution**: Model the tutor as an "agent" in an environment. Use Reinforcement Learning (RL) to let the agent learn the best policy over thousands of simulated sessions.
- **Tool**: gymnasium

ML for the Tutoring Model: Dialogue Systems

- **Problem**: Our tutor can't have a conversation. It can only give quiz questions
- **ML Solution**: Use a pre-trained conversational LLM to power a chatbot that can answer student questions in natural language.
- **Tool**: transformers

ML for the UI Model: Affective Computing with *TextBlob*

- **Problem**: The system is unaware of the student's emotional state. A frustrated student may be about to quit
- **ML Solution**: Use Natural Language Processing (NLP) to perform sentiment analysis on student input to detect frustration or other emotions. This is a form of affective computing
- **Tool**: TextBlob, a user-friendly Python library for common NLP tasks

Reminder of ML Aspects Covered Before

- Domain
  - Automatic Skill Discovery
  - Difficulty Estimation
  - Knowledge Tracing
- Student
  - Prediction
  - Classification
  - Recommendation
  - Clustering
- Tutoring
  - Policy Learning
  - Dialogue Systems
  - Feedback Personalization
  - Adaptive Hinting
- UI
  - Adaptive Feedback Presentation
  - Affective Computing
  - Personalization

Part 3: Live Demo

THE STUDENT JOURNEY

Additional Tools & Resources

- Domain Model (The Rulebook)
  - Neo4j: A graph database for modeling complex skill dependencies.
  - RDFLib: A Python library for creating formal knowledge representations (ontologies).
- Student Model (The Scorecard)
  - SQLite: A simple, file-based database (built into Python) to manage multiple students.
  - pyKT: A library for more advanced Deep Learning Knowledge Tracing models.
- Tutoring Model (The Game Master)
  - Rasa: An open-source framework for building conversational AI and chatbots.
  - Gymnasium (formerly OpenAI Gym): A toolkit for developing Reinforcement Learning policies.
- UI Model (The Game Console)
  - Plotly Dash: A Python framework for building interactive dashboards and data visualizations.
  - Django: A full-featured Python web framework for building large, robust applications.
- Project Management
  - Git & GitHub: The industry standard for version control and team collaboration.

Public Datasets for Your Project

- Mathematics
  - MATH Dataset: Challenging competition math problems with step-by-step LaTeX solutions.
  - GSM8K: Grade-school math word problems with reasoning and final answers.
- Science & General Knowledge
  - SciQ: Science exam questions with multiple-choice answers and support paragraphs that serve as explanations.
  - OpenBookQA: Science questions based on a set of open-book facts.
- Programming
  - Project CodeNet: A massive dataset of programming problems and millions of user-submitted solutions.
  - APPS: A dataset of programming problems with test cases for validation.
- Pro-Tip for Your Project:
  - For the scope of a class project, creating your own small, high-quality dataset is often better than trying to parse a massive public one. This gives you full control over the content and is a key part of modeling your domain.

Your Project Roadmap & Final Advice

- **The Foundation**
  - Pick a **very small domain** (e.g., 3-4 related concepts).
  - Build your Domain Model using **JSON files**.
  - Create a simple **Command-Line Interface (CLI)** for your UI.
- **The Logic**
  - Implement a rules-based Tutoring Model with if/else logic.
  - Build the basic Student Model to track scores in a JSON file
- **Connect & Test**
  - Integrate all four components.
  - Your goal is to have a complete, working, rules-based tutor.
- **After: Upgrade with ML**
  - Pick one component and upgrade it using one of the ML techniques we discussed.

Questions?