Experiment: Building a Mood Detector

Rules vs. Machine Learning

In this experiment, you build a mood detector that predicts a person's mood from what they write in a chat. You start with a rule-based approach, explore its limitations, and then build a second version using a machine learning approach. The goal is to compare both approaches and reflect on their strengths and weaknesses.

Task 1: Project Setup

In this experiment, you write some code. Let's begin by setting up your project environment.

1. Make sure your Master Brick successfully connects to your computer and that you can control the LED using the Brick Viewer.
2. Create a script called `mood_rb.py`. This script will be a Python program that connects to the LED. Set the LED to *off* initially.
3. After initializing the LED, add a loop that continuously reads text input from the user. After the user types a message and hits ENTER, print the message back to the console. If the user enters `bye`, exit the program.

Task 2: Rule-based Mood Detection

In this task, your code analyzes the user's input using a simple set of rules to determine the user's mood.

4. Develop a concept for how you want to determine the mood from the text input. For example, you could look for keywords that indicate a good mood (e.g., "happy", "great", "awesome") or a bad mood (e.g., "sad", "terrible", "angry"). Write down your ideas for the rules you want to use.
5. Implement a first version of your mood detector in Python using the rules you defined.
    - If the detector predicts a good mood, turn the LED green.

- If the detector predicts a bad mood, turn the LED red.
- If it cannot decide (no rule matches), turn the LED blue (neutral/unknown).

What programming constructs do you need to implement your rules?

6. Test your mood detector with the following sentences and observe how it behaves:

   a. Not bad at all, actually.
   b. Well, that could have gone worse.
   c. I guess this is fine.
   d. Oh great, another problem.
   e. I'm not unhappy with the result.
   f. Fantastic… now it's broken again.
   g. I was worried, but now it seems okay.
   h. That's just perfect.
   i. I can live with that.
   j. It's working, though I don't feel great about it.

7. Extend your mood detector with an additional label:

   - Hungry mood: If the user message contains keywords that indicate hunger or craving for food, turn the LED yellow.

8. Test your rule-based detector with additional inputs and edge cases (e.g., negation, sarcasm, mixed emotions, ambiguous statements). Can you find examples where it fails to correctly identify the mood? Write down your observations.

Task 3: Learning-based Mood Detection

Now test whether a machine learning approach can do better than your rule-based system. You will integrate a large language model into your program to analyze the user's input and predict the mood.

9. Create a new script called `mood_ml.py` as a copy of `mood_rb.py`. Remove the code that implements the rule-based mood detection.

10. Add the necessary code to connect to the OpenAI API and use a language model to analyze the user's input. You can use the `openai` Python library (`pip install openai`). You can obtain an API key from your instructor.

11. Create a prompt that instructs the model to classify the user's message into one of the following labels: `good`, `bad`, `neutral`, `hungry`. Send the user's message to the model and receive its response.

12. Based on the model output, set the LED color accordingly (green = good, red = bad, blue = neutral, yellow = hungry). How can you make the model output consistent and easy to parse?

Task 4: Systematic Testing and Comparison

13. Create a table (e.g. Excel) with at least 20 test sentences and assign each one an expected mood label. Include straightforward cases as well as difficult cases such as negation, sarcasm, mixed emotions, and ambiguous statements.
14. Test both the rule-based and the learning-based system on all examples. Automate this process and record the predictions of both systems and print the accuracy to the console.
15. Extend both systems so they also provide a short explanation of their decision. Compare how understandable and trustworthy these explanations are. Does it make a difference which language model you use?
16. Modify the learning-based system so that the model returns the labels `good`, `bad`, `neutral`, or `hungry` as a *structured JSON output*. Why is this useful when integrating the model into a program?

Task 5: Reflection and Discussion

Reflect on the following questions and write down your thoughts. Discuss with your peers:

17. Where did the rule-based system work well, and where did it fail?
18. In what way did the learning-based system behave differently?
19. Which system was easier to build?
20. Which system is easier to understand and debug?
21. Which system would you trust more in a real application, and why?
22. What does this experiment show about the difference between rules and learning?