CS 59000 – Expert System

Fall 2022

DISEASE PREDICTION SYSTEM USING BACKWARD CHAINING

[*]Likhita Budhavaram[a]

[a] Department of Computer Science, Purdue University, Fort Wayne

Fort Wayne, Indiana, 46805

Under the guidance of

Professor Mehrdad Hajiarbabi

Department of Computer Science, Purdue University, Fort Wayne

Fort Wayne, Indiana, 46805

Abstract

People may have second thoughts concerning events that occurred regarding their physical condition. Especially when there is nowhere to inquire about the suffering they are feeling. As a result, people frequently ignore and undervalue the symptoms they are having. Most people are aware that simple treatments are not necessary for mild ailments like colds or diarrhea. The issue is that a seemingly unimportant ailment could be a sign of a major condition. As a result, in order to diagnose the signs of a disease, one must be aware of its symptoms. The application offers advice on what to do when someone experiences these symptoms, including whether they should seek immediate medical attention. This will help them diagnose the symptoms they are experiencing. This application employs the backward chaining method of inference. Using this application, disease diagnosis results can be determined through the consultation process or by accurately and speedily responding to the questions presented by the system.

Introduction

The study of expert systems examines how expert thought processes are incorporated into computer technology. One technique that is used to choose an expert system is to trace backwards or (backward chaining). This technique can be used in expert systems to discover and diagnose problems. Backward chaining is a type of verification based reasoning that begins with a conclusion (goal-driven). This approach is effective for resolving issues that are modeled as structured issues. To choose wisely from a wide range of options is the aim of this inference.A diagnosis is one of the choices that may be made using an expert system extremely well.

Two important parts of building an Expert System:
1. Knowledge Acquisition-First step is knowledge acquisition gather/extract relevant knowledge. It is a crucial activity in the learning cycle since it aids in an organization's ongoing development and expansion of its knowledge base.

2. Knowledge Engineering-Knowledge engineering which means build the expert system knowledge base. A branch of artificial intelligence (AI) known as knowledge engineering seeks to replicate the thinking and actions of a human expert in a particular field. Expert systems are created using knowledge engineering technologies to help with problems pertaining to their preprogrammed field of knowledge.

Expert System Schematic

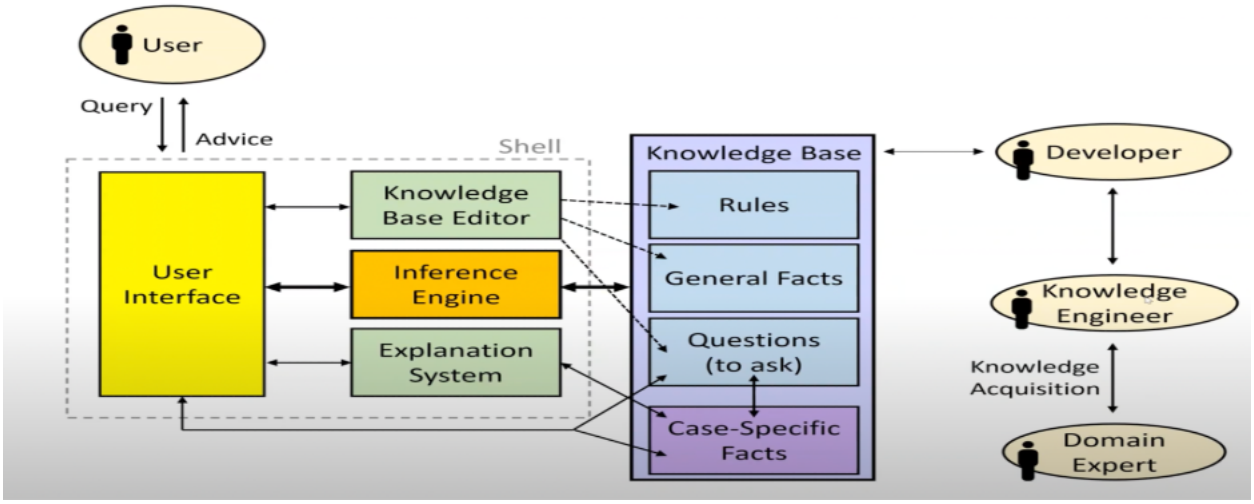


Fig 1: Schematic representation of Expert System

Domain Expert

Anyone can be considered domain expert if he or she has deep knowledge (rules and facts) and strong practical experience in particular domain.

Knowledge Engineer

Someone who is capable of designing, building, and testing an expert system.

Knowledge Acquisition

Extraction and formulation of knowledge derived from various sources, especially from human experts.

Inference engine

Component of the system that applies logical rules to the knowledge base to deduce new information.

Knowledge base

An artificial intelligence knowledge base tries to compile human expert information to aid in decision-making, problem-solving, and other processes. Knowledge base systems have been created over time to serve a variety of organizational operations.

The purpose of the Knowledge Acquisition and Learning Module is to enable the expert system to continuously learn new information from a variety of sources and store it in the knowledge base. At this part we will incorporate our rules, facts and question files that helps the system to use.

User Interface: With the help of this module, a non-expert user can communicate with an expert system and solve an issue.

Explanation Module: This module aids the expert system in explaining to the user how it arrived at a specific decision.

Implementation

The project was implemented using backward chaining and pyke. By offering a knowledge-based inference engine (expert system) implemented entirely in Python, Pyke introduces Logic Programming (influenced by Prolog) to the Python community.Unlike Prolog, Pyke is integrated with Python, enabling you to use Python expressions and statements within your expert system rules. Pyke was created to raise the bar for code reuse dramatically.

This is how it goes:

1. You create a series of Python functions as well as a set of Pyke rules to control how these functions are combined and configured.
2. Within the function body, these functions make reference to Pyke pattern variables.
3. Pyke might use different constant values for each of the pattern variables used in the function body when instantiating each of your functions several times.
4. These all emerge as different functions in different situations. To address a particular purpose or use situation, Pyke then automatically assembles these modified functions into a full program (function call graph). Pyke refers to this call graph function as a plan.

In this approach, Pyke offers a mechanism for you to drastically modify and adapt your Python code for a particular use case or purpose. By doing this, Pyke becomes effectively a very high-level compiler. Additionally, adopting this strategy results in notable improvements in performance.And Pyke succeeds well in doing this, offering orders of magnitude gains in:

1. Code flexibility (or customization)
2. Performance
3. Code Reuse

Pyke does not replace Python, nor is meant to compete with Python. Python is an excellent general purpose programming language, that allows you to "program in the small". Pyke builds upon Python by also giving you tools to directly program in the large

Steps to implement pyke

Step 1: Create an engine object

Loading knowledge_engine framework

- From pyke import knowledge_engine
- my_engine=knowledge_engine.engine(__file__)

Step 2: Activate rule bases

- my_engine.activate(‘rule_base_file’)

Step 3: prove goals

- From pyke import goal
- my_goal=goal.compile(‘goal’)

The following screenshot shows python code that loads framework and calls rules and question bases.

```
import contextlib
import sys

from pyke import knowledge_engine
from pyke import krb_traceback

engine = knowledge_engine.engine(__file__)

engine.reset()
engine.activate('rules')

try:
    with engine.prove_goal('rules.Disease_Prediction($d1,$d2)') as gen:
        for vars, plan in gen:
            print("You are suffering from %s and recommended to take %s tablet" % (vars['d1'], vars['d2']))
except Exception:
    # This converts stack frames of generated python functions back to the
    # .krb file.
    krb_traceback.print_exc()
    sys.exit(1)


print("done")
#engine.print_stats()
```

Fig 2: Python file that imports knowledge engine

Pyke has three different kinds of source files for the three main types of knowledge bases:

1. Knowledge Fact Base (KFB) files for fact bases.
2. Knowledge Rule Base (KRB) files for rule bases.
3. Knowledge Question Base (KQB) files for question bases.

Each type of source file ends in a different file suffix:.kfb, .krb or .kqb. Place all these source files into a directory structure. Then include this directory as an argument to the knowledge_engine.engine constructor. This will recursively search your directory for these three types of source files, compile them, and load them into the engine. How you organize these files into subdirectories is up to you -- the directory structure does not matter to Pyke.

1. **Rule base:** Stores rules in the form of knowledge rule base(.krb) file.
A single rule base may contain both forward chaining rules and backward chaining rules.
Rules have two parts:

- If part (containing a list of statements called the premises)
- Then part (containing a list of statements called the conclusions)
- Each of these if and then parts contain one or more facts or goals.

    Syntax: If [A],[B], and [C], Then [D] and [E]

    Semantics: If A,B, and C are true, then D and E are true

```
# rules.krb

Shortness_Breath_check
    use symptom1(Shortness_of_breath)
    when
        questions.any_EarlySymptoms($symptom1)
        check $symptom1 in (1,)

Tight_Chest
    use Symptom2(Tight_Chest)
    when
        symptom1(Shortness_of_breath)
        questions.any_bodypains($Symptom2)
        check $Symptom2 in (1,4,)

Wheezing
    use check_SevereSymptoms(Wheezing)
    when
        Symptom2(Tight_Chest)
        questions.check_SevereSymptoms($Symptom3)
        check $Symptom3 in (1,)

Disease_Prediction1
    use Disease_Prediction(Asthma,Albuterol)
    when
        check_SevereSymptoms(Wheezing)
```

In order to make the rule succeed, Pyke tries to match all statements with facts within the if clause through a process called backtracking.

2. **Question base:** Stores questions for end users in the form of knowledge question base(.kqb) file.The .kqb file contains all the information about the question needed to ask the question, validate the answer, and output the appropriate review text.

```
# questions.kqb

any_EarlySymptoms($symptom1)
    Is the patient currently suffering from any of these Early symptoms?
    ---
    $symptom1 = select_1
        1: Shortness of breath
        2: Cough
        3: Yellow Mucus
        4: Running Nose
        5: Rash on neck
        6: Fever or chills

any_bodypains($Symptom2)
    Is the patient currently suffering from any of these body aches?
    ---
    $Symptom2 = select_1
        1: Chest Pain
        2: Throat Pain
        3: Heart Pain
        4: Tight Chest
        5: Ear infections
        6: Muscle aches

check_SevereSymptoms($Symptom3)
    Is the patient currently suffering from any of these Severe Symptoms?
    ---
    $Symptom3 = select_1
        1: Wheezing
        2: Constant fever
        3: Rapid Heart beat
        4: Sneezing
        5: Blindness
        6: Loss of taste and smell
```

Working Process

When Pyke receives a question from your program, backward chaining rules are applied (i.e., asks Pyke to prove a specific goal). Pyke will only do the proof using activated rule bases.

Pyke searches for rules whose then component fits the objective to do backward chaining (i.e., the question). Once such a rule is discovered, it attempts to (recursively) verify all of the subobjectives in the if portion of that rule. In addition to being subgoals for other backward-chaining rules, several of these subgoals are matched against facts. The rule is successful and the primary objective is established if all of the subgoals can be demonstrated. If the rule is violated, Pyke attempts to identify another rule whose then part satisfies the objective.

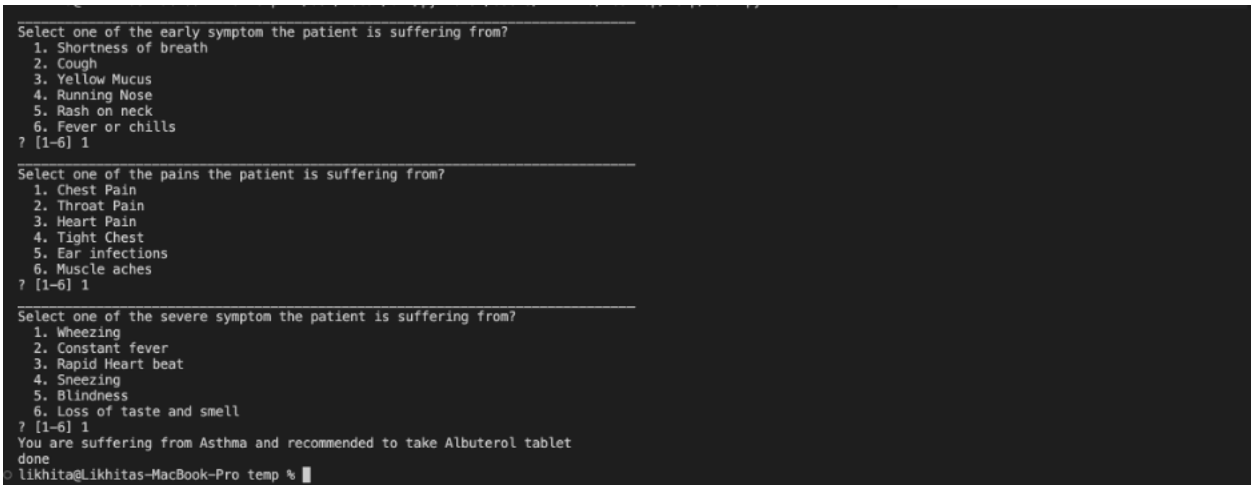
So Pyke ends up linking (or chaining) the if part of the first rule to the then part of the next rule. Pyke begins by identifying a rule whose then part corresponds to the objective. The if portion of the rule is then processed by Pyke. It could connect (or chain) to the following clause in another rule. It is known as backward chaining because Pyke executes these rules in the opposite order from which we often think of utilizing rules—from then to if to then to if. Pyke instructs you to write your backward-chaining rules upside down by writing the then part first (as that is how it is processed) in order to make this more obvious.

But then-if rules sound confusing, so Pyke uses the words **use** and **when** rather than **then** and **if**. You can then read the rule as "use" this statement "when" these other statements can be proven.

The backtracking method states that "success goes down (or ahead) and failure goes up (or back)." consists of the full chain of inference that begins with the attempt to verify the top-level aim rather than just the steps inside a single rule's when clause. Python and other conventional programming languages don't support this execution mechanism. Backward-chaining systems' strength comes from their ability to try alternative solutions by going back to any point in the calculation.

Result

The screenshot includes the result of the system. The code runs and user is asked to answer three questions and choose one answer among the options. On selection the programs runs and through backward chaining, the symptoms are read, and the disease is predicted. The output will be the disease that the user is facing and medication that he has to take in order to cure them.



References

1. https://pyke.sourceforge.net/
2. https://www.healthline.com/health/airborne-diseases#types
3. Expert System: Vertigo Disease Diagnosis with Backward Chaining Method
    a. [DOI:10.1109/TSSA51342.2020.9310820]