Expert System for Diagnosis of Respiratory Diseases

Radu Potop
Masterat TI, Anul II

Abstract

This paper follows the process of designing and implementing an Expert System. We will call this system ES, for a lack of a better name. The restricted field which this system will explore are respiratory diseases. This system uses rules and Certainty Factors to calculate the probability of a certain disease. The system is implemented in Python and runs as a web application - trying to be a more modern approach to expert systems.

Introduction

An Expert System is a software application that consults an existing knowledge base [1] to come up with new conclusions about a specific domain or with new solutions to a problem. Expert systems are often used to aid human experts in making better decisions.

Expert Systems focus on restricted areas of expertise and help solve problems.

The expert system described in this paper focuses on diagnosing respiratory diseases. Of course the database on which the system engine operates can be changed to other fields, but some entries, such as Certainty Factors need to be adjusted. The implementation is more of a proof of concept, but it works well with the default diseases, symptoms and rules. The purpose of this paper was to follow the steps that allow creation of an expert system, from the design phase, to the implementation and testing. The system is rule based and used Certainty Factors (CFs) and is quite simple in concept.

Design and Development

The database

The first step in designing the expert system is to design the database, which holds the actual data on which the system operates. The database needs to hold three main sets of data: diseases, symptoms and rules [2].

Diseases are the conclusions (the answers) to which the expert system arrives to after the initial questions are asked. Each disease has a set of symptoms that characterize that disease. All the symptoms of a disease can uniquely identify that disease. The number of default diseases given in the application are three.

The symptoms are part of the initial questions that are given to the end user. They help diagnose the disease and get to a conclusion. The number of default symptoms given in the application are ten.

The rules tie the symptoms to the diseases. The rules can be seen as a matrix (Fig. 1). Each disease has a number of symptoms. Each connection between a disease and a symptom is called a rule. Each rule has a certainty factor attached [3]. This represents the probability of having that disease if the symptom is True. The certainty factor is represented in percentages.

	Pneumonie	Amigdalita	Viroza respiratorie
Tuse	X		X
Durere in gat		X	
Amigdale inflamate		X	
Dureri piept	X		
Febra	X	X	X
Congestie nazala	X		X
Dificultate inghitire		X	
Dureri cap			X
Stranut	X	X	X
Frisoane	X	X	X

Fig. 1. A matrix of diseases and their symptoms

Using Fig. 1 we can design our database schema more easily. The model [4] of the database tables, with the necessary restrictions, would be described as following:

diseases:

id_disease | name

id_disease = PK

name = varchar

symptoms:

id_symptom | name

id_symptom = PK

name = varchar

rules:

id_rule | id_disease | id_symptom | cf

id_rule = PK
id_disease, id_symptom = FK
cf = int, this is the certainty factor

Of course this is a simple model that can be extended to contain users and a history of each user. But for simplicity we will not focus on those aspects.

The application

The application is the piece of code that operates on our data. This can be called an inference engine [2] to adhere to the expert systems terminology. The application extracts data from the database using SQL, performs operations on it and serves the result to the user.

For development of this application we use Python programming language. However we will not use raw Python, but use a web framework to ease the development of the application. The framework chosen was Webpy [5]. This framework gives us the tools to work with the database, templates and so on in a more efficient manner. However it does not help create the core engine of the expert system.

The application was designed on three layers. This is a well known design pattern known as Model - View - Controller (MVC) [6].

The Model contains the database logic and works with data. It exposes its functionality through custom functions made by the developer. Webpy helps us with this task by providing a Database Abstraction Layer (DAL) [7].

The View contains the application web interface. This is made from HTML files, rendered by the template system with data inserted from the Controller. The web interface is presented to the user. This is a series of check-boxes with symptoms that the user has to choose from.

There are basicailly two pages: the form page and the results page.

In the results page there are debug statements included that can help determine the system’s reasoning behind the decisions.

The Controller is the core, or the engine of our application. Here data is taken from the Model, manipulated and sent to the View to be rendered. The Controller also handles HTTP logic such as POST and GET.

The Controller gets the input from the View with the user's symptoms. It then searches through rules that have that symptom present. Remember that rules contain a relation between symptoms, diseases and CFs. For each disease where that symptom is present it creates an entry in a dictionary (a type of Python data structure). Then for that disease it stores the CF associated to the rule. It then proceeds to do this for other diseases where the symptom applies.

After this process the next symptom is taken from the user input and rules are searched for its presence. The CF values from the new found rules are added to the existing CF values in the dictionary, for each disease. This happens until all symptoms are handled.

The following pseudo-code represents the simplified functionality of the inference engine:

```
# Determine CF for each disease
FOR EACH symptom
    FOR EACH rule
        disease_id = initial_CF + CF
```

Then the application determines which disease has the maximum CF value and declares that disease the winning disease.

Further design and implementations details will not be covered in this paper, as it is not necessary. The source code can be obtained from Bitbucket, using Mercurial [8]. The code can be studied to fully understand the application.

Deployment

Our application was deployed on the CherryPy web server. This is a special web server that knows the WSGI protocol used by Python web frameworks.

The application only needs the necessary Python modules present: webpy and mysql-python. Then it can be launched from the command line using the command:

```
./es.py
```

Conclusions

The designed application we designed and develop can help us better understand how an expert system works. This is mostly a concept, but it works satisfactory well.

Because each disease has unique symptoms and the CF values are properly balanced, the system gives satisfactory results and it can accurately diagnose the three diseases included in the default data.

References

1. Doru Adrian Panescu, Sisteme bazate pe cunostinte, Cap 1.5, http://www.ac.tuiasi.ro/ro/library/SBCHTML/curs/Cap1.5.pdf
2. Madalina Roxana Buneci, Sisteme Expert Bazate Pe Reguli, Cap 8, http://www.utgjiu.ro/math/mbuneci/book/exp/cap08.pdf
3. Edward Shortliffe, Certainty Factors: MYCIN and Inexact Reasoning, http://www.computing.surrey.ac.uk/ai/PROFILE/mycin.html#Certainity Factors
4. F. Radulescu, Modelul Entitate-Asociere clasic, http://scad.cs.pub.ro/pc1/erclasic.pdf
5. Webpy framework, http://webpy.org/
6. Trygve Reenskaug, Models - Views - Controllers, http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf
7. Chris Baron, Massimo Di Pierro, Publishing Linked Data Using web2py, http://web2py.com/semantic/static/semantic.pdf
8. Source code, https://bitbucket.org/wooptoo/es