A knowledge-based system for the underwriting process

AI Technologies Part 1 Group Work

Module:	AI Technologies - Part 1
Authors:	Ulrich Pogson
	Adam Threlfall
Lecturers:	Prof. Dr. Holger Wache & Dr. Stephan Jüngling
Place, date:	Basel, 29.10.2021

Table of contents

Table of contents **1**

Introduction **2**

Documentation **2**

Knowledge from the experts 2

Decision Table 2

Decision Table Overview 3

Single Decision Tables 3

Prolog 4

Version 1 4

Version 2 5

Version 3 6

Knowledge from past applications 7

Combination of knowledge from experts and past applications 8

Introduction

This is a brief documentation on how we tried to solve the group assignment in the course "AI Technologies Part 1" lectured by Prof. Dr Holger Wache and Dr. Stephan Jüngling.

The assignment description to this group work was the following:
"A small health insurance company wants to standardize its underwriting process. In this process an applicant applies for a health insurance company. But different workers decide differently (i.e. based on their own knowledge) and therefore the acceptance depends highly on which worker gets the application on their desk. The insurance company now decides to collect the knowledge and define some guidelines that will be obligatory in future. In order to support them the insurance company also decided to support the workers by a knowledge-based system which gives them some recommendations on how to decide in a particular case.

There are two kinds of sources for such knowledge. The first sources are the experts themselves. The second sources are the past applications stored in a sheet. The company wants to extract the knowledge out of that data as well."

Our task was to build a knowledge-based system, which involved two steps:

1. To transform the knowledge from experts into a format which can be used by an expert system, i.e. model that knowledge into an appropriate knowledge representation language.
2. To extract the knowledge from the sheet and translate it also into the knowledge representation formalism.

The aim of this document is to provide an insight on the process and reasoning of how we tried to achieve these two tasks.

Documentation

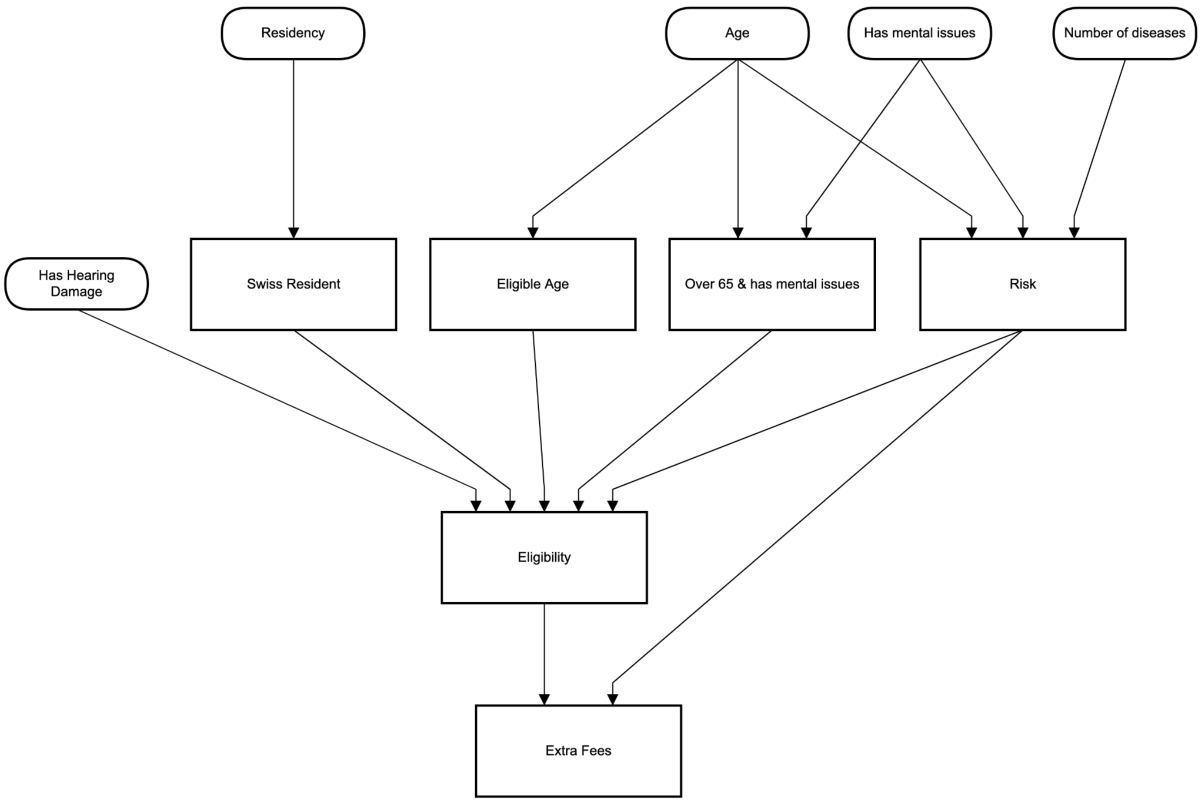
Knowledge from the experts

Decision Table

The decision tables were created based on the knowledge from the expert. Camunda Modeler was used to create the decision tables. It took a few iterations to achieve the final result. The first step was to figure out how best fit the different data into the tables. With each iteration, tables were shortened and new ones were created. During the PROLOG rule definition we had another iteration.

Download DMPN Decision Table

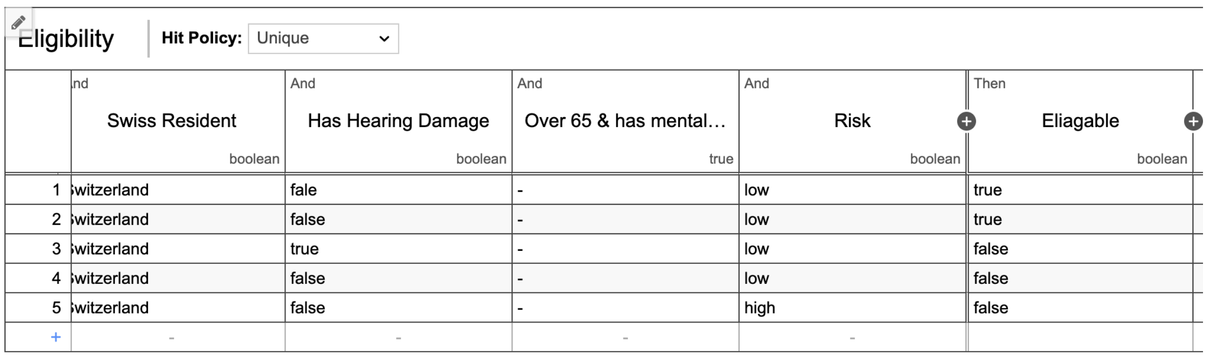
Decision Table Overview



DMN Decision Table Overview created in Camunda Modeler by Ulrich & Admin

Single Decision Tables

For Swiss Resident, Eligible Age, Over 65 & has mental issues, Eligibility and Extra Fees a decision table was created. Below is an example for the Eligibility decision table. The rest of the tables can be found in the DMN model.



Eligibility | Hit Policy: Unique

	nd	And	And	And	Then
	Swiss Resident	Has Hearing Damage	Over 65 & has mental…	Risk	Eliagable
	boolean	boolean	true	boolean	boolean
1	witzerland	fale	-	low	true
2	witzerland	false	-	low	true
3	witzerland	true	-	low	false
4	witzerland	false	-	low	false
5	witzerland	false	-	high	false
+	-	-	-	-	

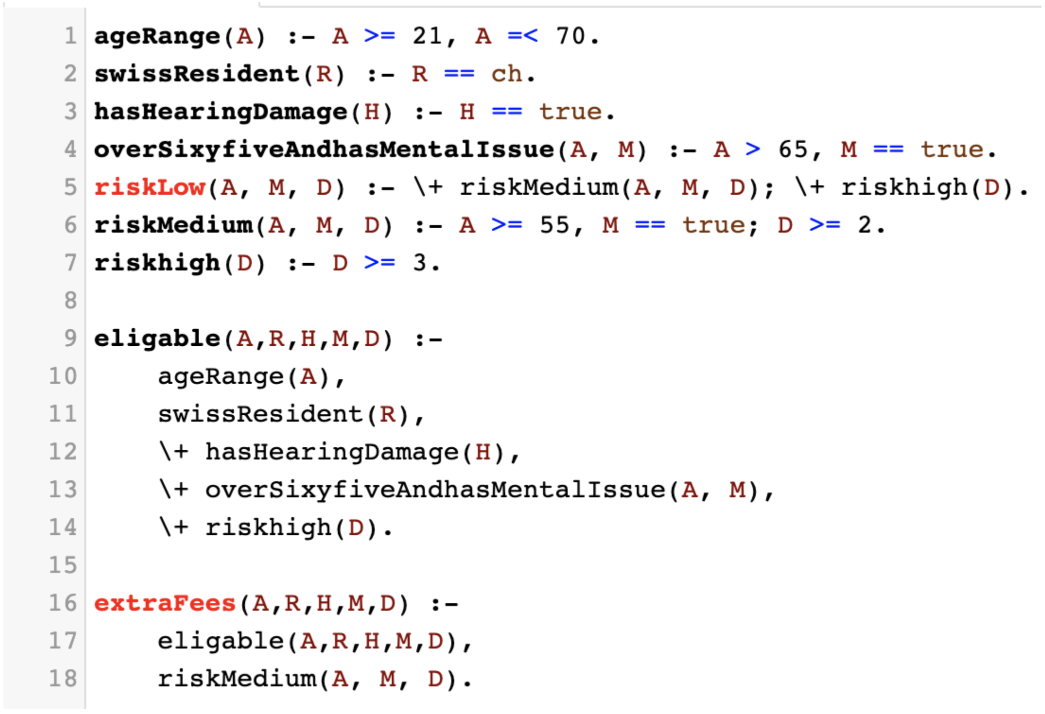
Decision table for Eligibility created by Ulrich & Adam

Prolog

In the Prolog code there are two main rules to be used. The eligible rule checks if the person is eligible and the second rule is used to check if the person has to pay extra fees.

Version 1

In the first version the value of the risk was defined in the predicate name.

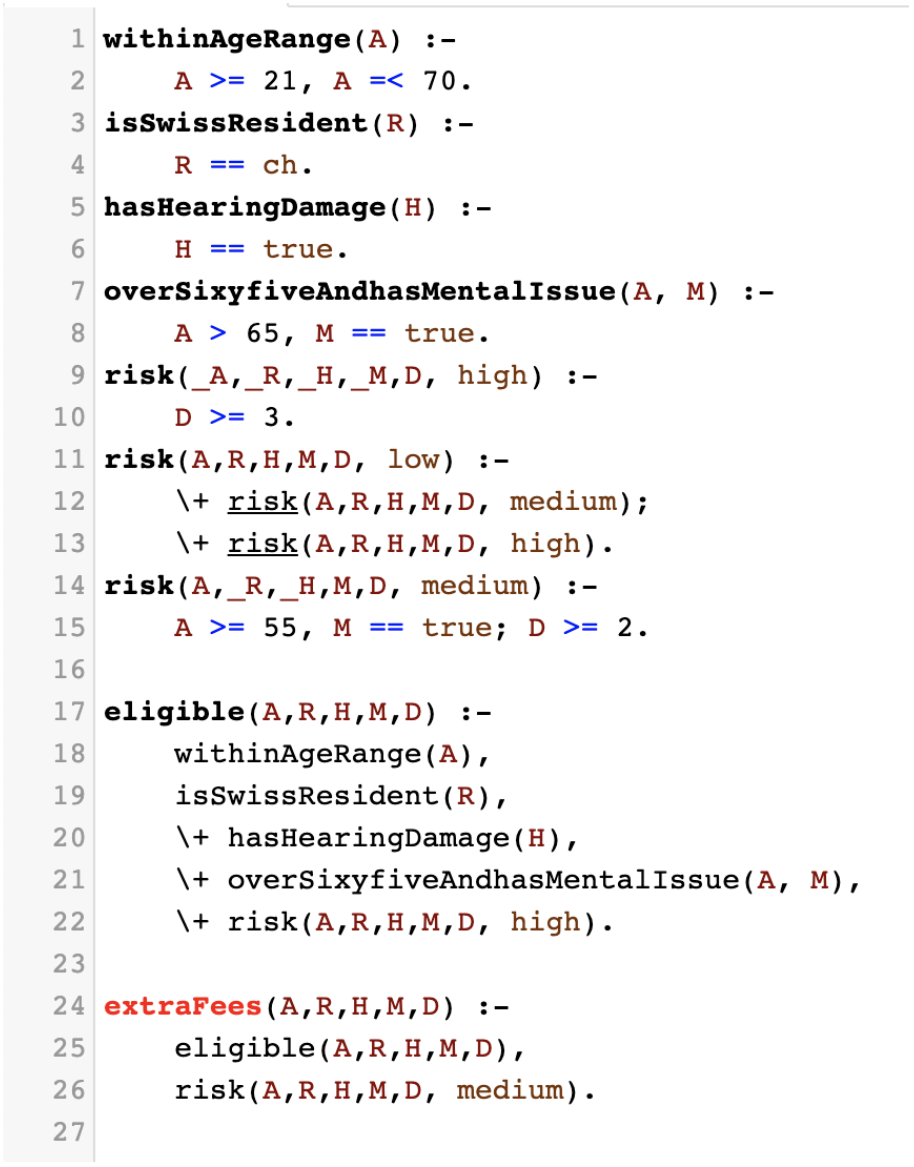


```prolog
ageRange(A) :- A >= 21, A =< 70.
swissResident(R) :- R == ch.
hasHearingDamage(H) :- H == true.
overSixyfiveAndhasMentalIssue(A, M) :- A > 65, M == true.
riskLow(A, M, D) :- \+ riskMedium(A, M, D); \+ riskhigh(D).
riskMedium(A, M, D) :- A >= 55, M == true; D >= 2.
riskhigh(D) :- D >= 3.

eligable(A,R,H,M,D) :-
    ageRange(A),
    swissResident(R),
    \+ hasHearingDamage(H),
    \+ overSixyfiveAndhasMentalIssue(A, M),
    \+ riskhigh(D).

extraFees(A,R,H,M,D) :-
    eligable(A,R,H,M,D),
    riskMedium(A, M, D).
```

Prolog Code created by Ulrich & Adam

Version 2

In the second version the value for the risk is moved to the arguments and removed from the predicate name. It is not ideal how the number of diseases has to be passed. This was attempted to be changed in version 3 but unfortunately was unsuccessful.



```prolog
withinAgeRange(A) :-
    A >= 21, A =< 70.
isSwissResident(R) :-
    R == ch.
hasHearingDamage(H) :-
    H == true.
overSixyfiveAndhasMentalIssue(A, M) :-
    A > 65, M == true.
risk(_A,_R,_H,_M,D, high) :-
    D >= 3.
risk(A,R,H,M,D, low) :-
    \+ risk(A,R,H,M,D, medium);
    \+ risk(A,R,H,M,D, high).
risk(A,_R,_H,M,D, medium) :-
    A >= 55, M == true; D >= 2.

eligible(A,R,H,M,D) :-
    withinAgeRange(A),
    isSwissResident(R),
    \+ hasHearingDamage(H),
    \+ overSixyfiveAndhasMentalIssue(A, M),
    \+ risk(A,R,H,M,D, high).

extraFees(A,R,H,M,D) :-
    eligible(A,R,H,M,D),
    risk(A,R,H,M,D, medium).
```

Prolog Code created by Ulrich & Adam

Version 3

There seems to be a better way using customer id and defining their disease: “disease(124, 'Mental Issue').”

```prolog
% Information regarding customer with ID 123.
age(123,65).
residency(123,ch).
disease(124, 'Mental Issue' ).
disease(124,'Hearing Damange').

withinAgeRange(ID) :-
    age(ID,Age), Age >= 21, Age =< 70.
isSwissResident(ID) :-
    residency(ID,ch).
hasHearingDamage(ID) :-
    disease(ID,'Hearing Damange').
overSixyfiveAndhasMentalIssue(ID) :-
    age(ID,Age), Age > 65, disease(ID, 'Mental Issue' ).
risk(_ID,D, high) :-
    D >= 3.
risk(ID,D, low) :-
    \+ risk(ID,D, medium);
    \+ risk(ID,D, high).
risk(ID,D, medium) :-
    age(ID,Age), Age >= 55, disease(ID, 'Mental Issue' ); D >= 2.

eligible(ID,D) :-
    withinAgeRange(ID),
    isSwissResident(ID),
    \+ hasHearingDamage(ID),
    \+ overSixyfiveAndhasMentalIssue(ID),
    \+ risk(ID,D, high).

extraFees(ID,D) :-
    eligible(ID,D),
    risk(ID,D, medium).

```

Prolog Code created by Ulrich & Adam

Knowledge from past applications

Colab Link

The knowledge gained from the classifications of past applications can be used to train a neural network which can be used for the classification of future applications. In order to do this a neural network needs to be modelled, trained and tested on sample data with known classification results.

The first step is to load the ‘project_data.csv’ data file into a ‘pandas’ table in order to act as a data source for the neural network to interact with during training and testing. Before the neural network can interact with the neural network however some preprocessing needs to be done to the table to ensure the data is in the correct format for compatibility with the neural network.

The ‘data’ contains dtype strings which need to be converted into equivalent integer values; homogenous integer data is much easier to work with in the context of neural networks. The strings are replaced such that they have a direct mapping to an integer.
E.g ‘low’—> 0, ‘medium’ —> 1, ‘high’ —> 2.

It can be a good idea to test the validity of the structure of the data, this can be done by using ‘seaborn.pairpoint’.

Once the data has been converted to a homogenous dtype integer the ‘pandas’ table can be split into two separate ‘pandas’ tables; one containing the input data and the other containing the target classification.

With the preprocessing done it is possible to begin modelling the neural network using ‘keras’. Since there are three possible unique outputs then the output layer must contain three unique nodes; one for each possible classification. The input layer contains seven nodes and the hidden layer was decided to have 100 nodes. The number of nodes in the hidden layer was decided upon through progressive experimentation in order to maximise the accuracy of the trained neural network. The data was split into training samples and validation samples with a 80:20 split respectively. 50 epochs was determined to be a suitable period to train the neural network to maximise accuracy while minimising computing resources and time.

After the training has been completed the neural network is evaluated to check the accuracy and loss.

The neural network can now be used on any appropriate sets of data and can decide on the classification. The classification decided by the neural network will be in the integer form of the classification, this can then be mapped back to the string form of the classification.
E.g 0–>’low’, 1–>’medium’, 2–>’high’.

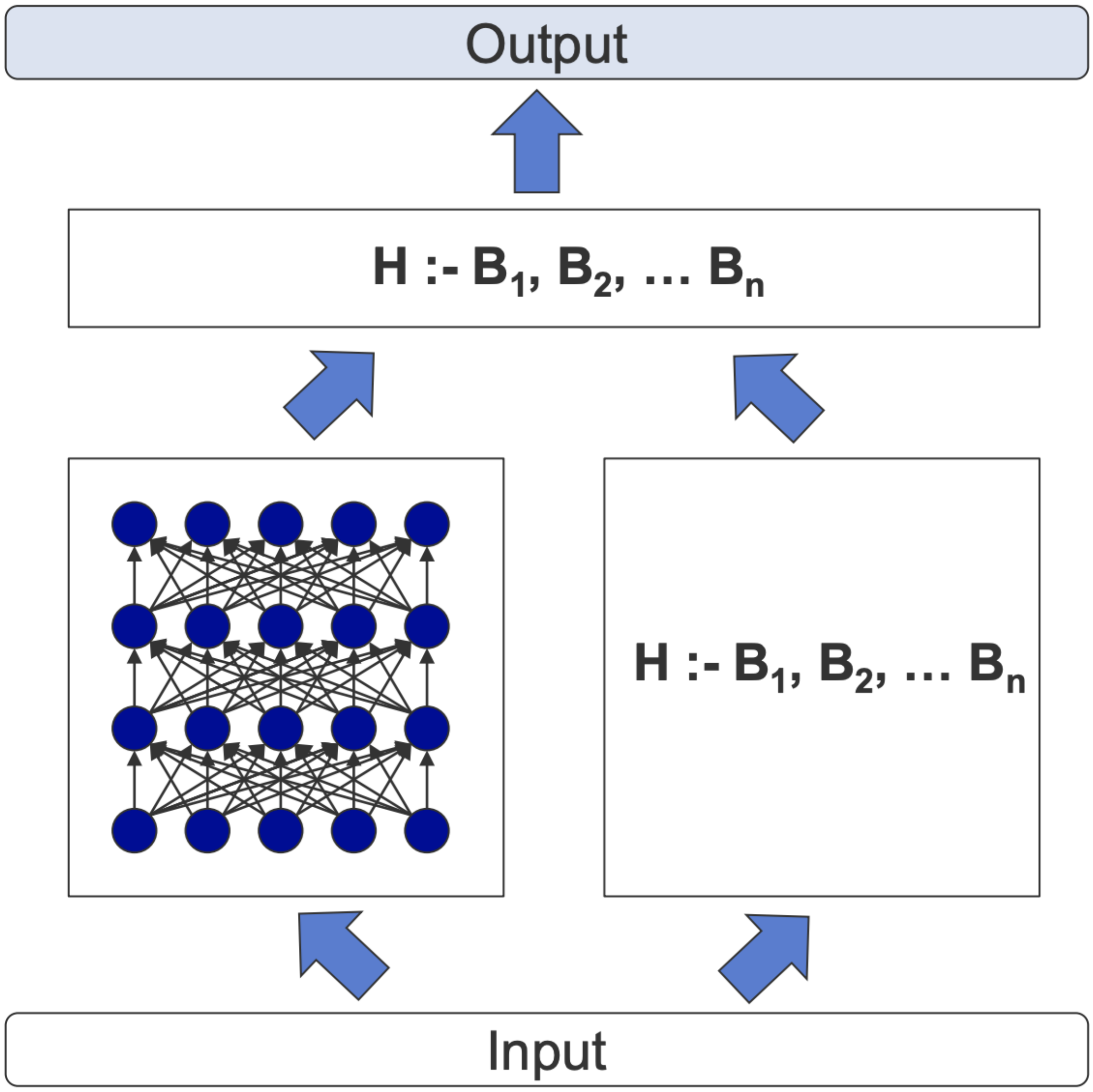
The training and test accuracy and loss can be easily visualised by plotting them as graphs of accuracy against epoch and loss against epoch respectively.

Combination of knowledge from experts and past applications

Even though arguments for the other options could also be found, it was decided that the “Parallel + Multiplexer (Prolog)” approach would be the best. For the input the data regarding the customer is provided. For the output we expect the risk level; “low”, “medium” and “high”. If the customer is ineligible then the output is “not-applicable”.

The decision table is unable to take into account all of the potential available data. The neural network however is able to process all potential available data and make a decision with it. This combines the knowledge from experts and past applications together.

The decision table in the second level choses the highest risk from both responses. By choosing the highest risk, the insurance company can prepare for the worst case scenario.



Source: Lecture slides, AIT AS2021 – Kombinging ML & KE - slide 51