Student: Diana Niti

Expert system for diabetes risk estimation

Nowadays, with the use of technology more and more processes are eased in all domains. After a careful analysis of the parts of processes which can be automated , specialists reached the conclusion that that improvements can be made in most departments such as engineering , economics , medical department etc. This project focuses particularly in the field of medicine , more specifically, in improving the time of interpreting analyses results. For such an operation usually a patient needs to consult the results with an expert, a specialist in Healthcare . However this dialog and first interpretation of results can be implemented with success using an Expert System. This System would allow the patient to complete a form that would inform him/her whether is necessary to make an appointment to a specialist or not.

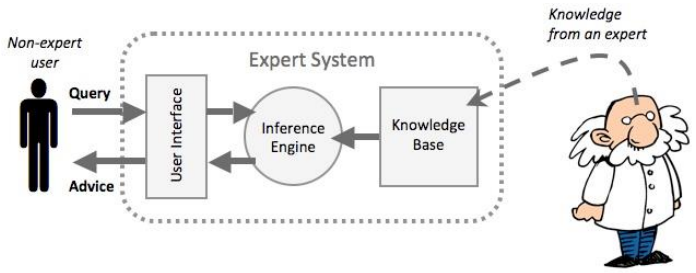
Human Expert	Expert System	Conventional Programs
Use knowledge in euristic form to solve problems in a narrow domain	Process knowledge expressed in the form of rules and use symbolic reasoning to solve problems in a narrow domain.	Process data and use algorithms , a series of well-defined operations, to solve general numerical problems.
In a human brain , knowledge exists in a compiled form	Provide a clear separation of knowledge from its processing	Do not separate knowledge from the control structure to process this knowledge.
Capable of explaining a line of reasoning and providing the details.	Trace the rules fired during a problem-solving session and explain how a particular conclusion was reached and why specific data was needed	Do not explain how a particular result was obtained and why input data was needed.

Comparison of expert systems with conventional systems and human experts.

In artificial intelligence, an expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning through bodies of knowledge, represented mainly as if–then rules rather than through conventional procedural code.

Expert systems were among the first truly successful forms of artificial intelligence. An expert system is divided into two subsystems: the inference engine and the knowledge base. The knowledge base

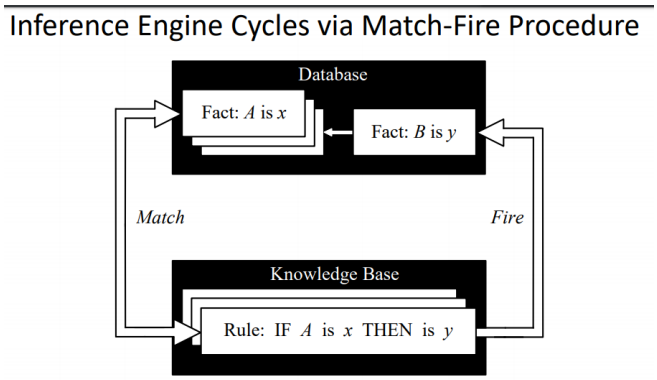
represents facts and rules. The inference engine applies the rules to the known facts to deduce new facts. Inference engines can also include explanation and debugging abilities.



Expert System Diagram

Software architecture

The inference engine is an automated reasoning system that evaluates the current state of the knowledge-base, applies relevant rules, and then asserts new knowledge into the knowledge base. The inference engine may also include abilities for explanation, so that it can explain to a user the chain of reasoning used to arrive at a particular conclusion by tracing back over the firing of rules that resulted in the assertion.



Inference Engine

There are mainly two modes for an inference engine: forward chaining and backward chaining. The different approaches are dictated by whether the inference engine is being driven by the antecedent (left hand side) or the consequent (right hand side) of the rule.

In forward chaining an antecedent fires and asserts the consequent. For example, consider the following rule:

$$R1 : Man(x) \implies Mortal(x)$$

A simple example of forward chaining would be to assert Man(Socrates) to the system and then trigger the inference engine. It would match R1 and assert Mortal(Socrates) into the knowledge base.

Backward chaining is a bit less straight forward. In backward chaining the system looks at possible conclusions and works backward to see if they might be true. So if the system was trying to determine if Mortal(Socrates) is true it would find R1 and query the knowledge base to see if Man(Socrates) is true. One of the early innovations of expert systems shells was to integrate inference engines with a user interface. This could be especially powerful with backward chaining. If the system needs to know a particular fact but does not, then it can simply generate an input screen and ask the user if the information is known. So in this example, it could use R1 to ask the user if Socrates was a Man and then use that new information accordingly.

For this project we have chosen forward chaining: We start from the facts and we deduct the result.

Library Experta: Python

Defining Knowledge base and inference engine

In the field of Artificial Intelligence, **inference engine** is a component of the system that applies logical rules to the knowledge base to deduce new information.

The knowledge base stored facts about the world.

The inference engine applies logical rules to the knowledge base and deduce new knowledge.

Fasting Blood Sugar Range	mg/dL	mmol/L
Hypoglycemia (low blood sugar)	30	1.7
	40	2.2
	50	2.8
	60	3.3
	70	3.9
Normal Blood Sugar	70	3.9
	80	4.4
	90	5.0
	100	5.5
* Pre-Diabetic Range	101	5.6
	110	6.1
	120	6.7
	125	6.9
* Diabetic Range	126	7.0
	140	7.8
	155	8.6
	160	8.9
	175	9.7
	190	10.6
	200	11.1
	250	13.9
	300	16.7
	400	22.2
	600	33.3

Chart of Glycemic levels

According to the diagram above we take into consideration that people who suffer from hyperglycemia have the range between 7 to 33.3 and above while on the other side people suffering from hypoglycemia have a range between 1.7 to 3.9.

Therefore for our system of rules and facts we would evaluate the patients with glycemic less than 4 as being suspect of Hypoglycemia , and for those whose blood tests indicates more than 7 we would put them into the risk of Hyperglycemia category. These 2 factors would raise the firsts concerns.

In addition to this we would take into consideration other parameters:

shakiness	>2. If more than two occurs the patient's risks of hypoglycemia increases
hunger	
sweating	

headache	
pale	
frequent urination	>2 . If more than two occurs the patient's risks of Hyperglycemia increases
thirst	
blurred vision	
headach	
drymouth	
smeling breath	
shortness of breath	

Implementing the table above in Python , using Experta Library.

The Rules and Facts:

```
@Rule(Personne(age=P(lambda x: x <= 99)))
def concerned_person(self):
    self.declare(Fact(concerned=True))
```

```
@Rule(Fact(concerned=True),
      Personne(glycemie=MATCH.glycemie))
def hyper_glycemy(self, glycemie):
    if glycemie > 7:
        self.declare(Fact(hyperglycemic_risk=True))
        print("Warning! High blood sugar")
    else:
        self.declare(Fact(hyperglycemic_risk=False))
```

```
@Rule(Fact(concerned=True),
      Personne(glycemie=MATCH.glycemie))
def hypo_glycemy(self, glycemie):
    if glycemie < 4:
        print("Warning! Low blood sugar")
        self.declare(Fact(hypoglycemic_risk=True))
    else:
        self.declare(Fact(hypoglycemic_risk=False))
```

As per the first part of the table: whether 2 or more signals of hypoglycemia are found it indicates low levels of sugar in the blood.

```
@Rule(Fact(concerned=True),
      AS.p << Personne(),
      TEST(lambda p: SUMFIELDS(p,
                               'shakiness',
                               'hunger',
                               'sweating',
                               'headach',
                               'pale') > 2))
def has_signs_low_sugar(self, p):
    self.declare(Fact(has_signs_low_sugar=True))
```

```
@Rule(Fact(concerned=True),
      Fact(has_diabetic_parents=True),
      Fact(has_signs_low_sugar=True))
def protocole_risk_low(self):
    print("Warning! The patient could be diabetic")
```

```
@Rule(Fact(concerned=True),
      Fact(hypoglycemic_risk=True),
      Fact(has_signs_low_sugar=True))
def protocole_alert_low(self):
    print("Alert! High risk of diabetes, you must see a doctor")
```

If the patient has at least one of his parents diabetic its also at risk of having diabetes.

```
@Rule(Fact(concerned=True),
      Personne(diabetic_parents=True))
def has_diabetic_parents(self):
    self.declare(Fact(has_diabetic_parents=True))
```

If more than two factors are present , the patient has signs of high sugar.

```
@Rule(Fact(concerned=True),
      AS.p << Personne(),
      TEST(lambda p: SUMFIELDS(p,
                               'urination',
                               'thirst',
                               'blurred_vision',
                               'headach',
                               'dry_mouth',
                               'smelling_breath',
                               'shortness_of_breath') > 2)
)
def has_signs_high_sugar(self, **_):
    self.declare(Fact(has_signs_high_sugar=True))
```

Also in the knowledge base the heredity plays an important part in this analysis.

```
@Rule(Fact(concerned=True),
      Fact(has_diabetic_parents=True),
      Fact(has_signs_high_sugar=True))
def protocole_risk_high(self):
    print("Warning! The patient could be diabetic")
```

```
@Rule(Fact(concerned=True),
      Fact(hyperglycemic_risk=True),
      Fact(has_signs_high_sugar=True))
def protocole_alert_high(self):
    print("Alert! High risk of diabetes, you must see a doctor")
```

Now the inference Engine operates on all this set of facts and rules in order to gather the results:

```
class InreferenceEngine(KnowledgeEngine):
    @Rule(Personne(age=P(lambda x: x <= 99)))
    def concerned_person(self):
        self.declare(Fact(concerned=True))

    @Rule(Fact(concerned=True),
          Personne(glycemie=MATCH.glycemie))
    def hyper_glycemy(self, glycemie):
        if glycemie > 10:
            self.declare(Fact(hyperglycemic_risk=True))
            print("Warning! High blood sugar")
        else:
```

Webpage Backend Node Express:

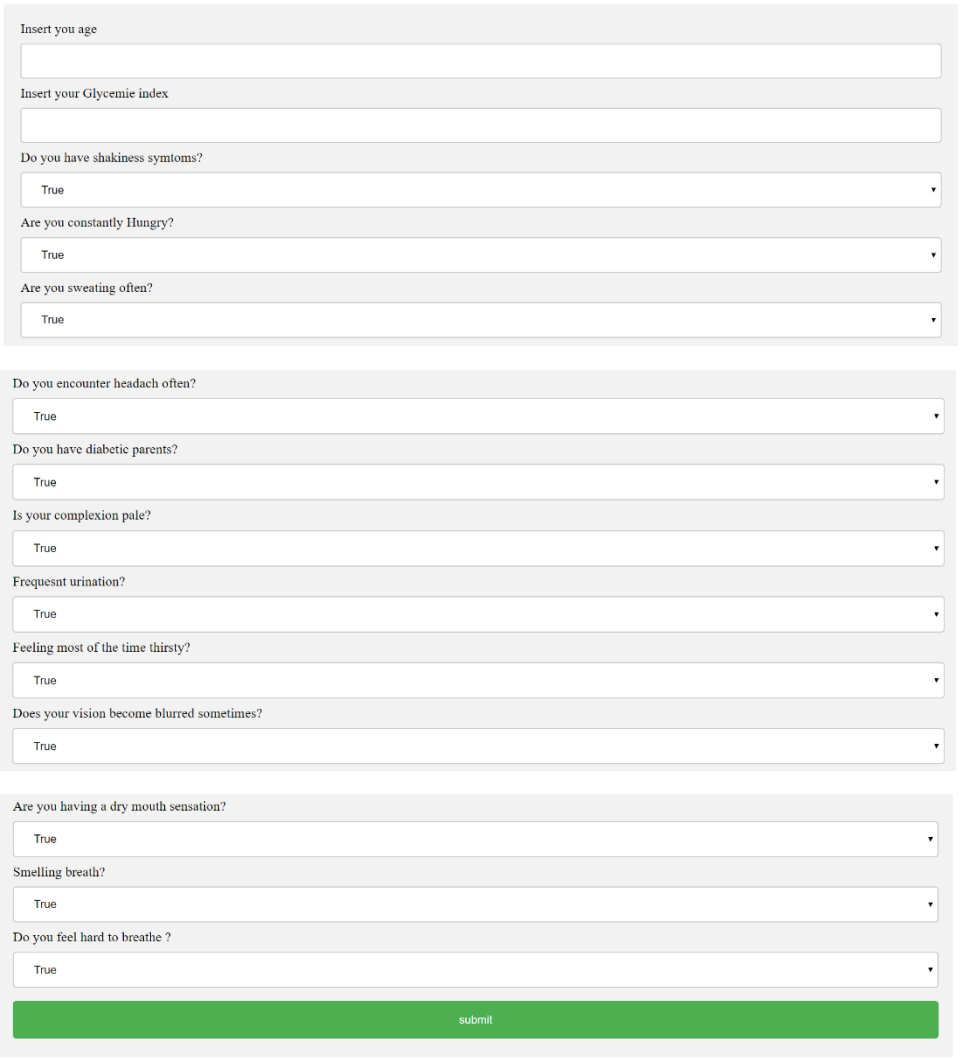
```
const express = require('express')

const app = express()

const PythonShell = require('python-shell');

var path=require('path');

app.get('/res/api', (req, res, next) => {
  let options = {
    mode: 'text',
    pythonPath: 'C:/Users/diana/AppData/Local/Programs/Python/Python36/python.exe',
    scriptPath: './',
    args: [req.query.age, req.query.glycemie, req.query.shakiness, req.query.hunger, req.query.sweating,
req.query.headach, req.query.diabetic_parents, req.query.pale, req.query.urination, req.query.thirst,
req.query.blurred_vision, req.query.dry_mouth, req.query.smelling_breath,
req.query.shortness_of_breath]
  };
  let details = {
    age: options.args[0],
    glycemie: options.args[1],
    shakiness: options.args[2],
    hunger: options.args[3],
```

```
    sweating: options.args[4],
    headache: options.args[5],
    parents: options.args[6],
    pale: options.args[7],
    urination: options.args[8],
    thirst: options.args[9],
    blurred_vision: options.args[10],
    dry_mouth: options.args[11]
  }
  PythonShell.run('expert-system.py', options, (err, results) => {
    if (err) throw err;
    // results is an array consisting of messages collected during execution
    console.log(results);
    res.json({ "res": results })
  });
});


  app.get('/',function(req,res){
  res.sendFile('index.html',{root: path.join(__dirname,'./')})
 // res.render('/res/api');
});
app.post('/res/api',function(req,res){
 let Age=age;
  let Glycemie=glycemie;
  let Shakiness=shakiness;
    let Hunger=hunger;
      let Sweating=sweating;
    let Headach= headach;
  let Diabetic_parents= diabetic_parents;
```

```
 let Pale= pale;

 let Urination=urination;

 let Thirst=thirst;

 let Burred_vision=blurred_vision;

 let Dry_mouth=dry_mouth;

 let Smelling_breath= smelling_breath;

  let Shortness_of_breaath=shortness_of_breath;

res.render('res/api',{

Age,Glycemie,Hunger,Sweating,Headach,Diabetic_parents,Pale,Urination,Thirst,Burred_vision,Dry_mou
th,Smelling_breath

})

});

app.listen(3000, () => console.log('Example app listening on port 3000!'))
```

Webpage frontend:

Medical Risk Estimation for Dianbetes



Results and remarks.

After simulating for a patient that have the following parameters :

Age:54

Glycemie:3

Shakiness: True

Hungry:False

Sweating:True

Headach:False

Diabetec parents: True

Pale:False

Frequent urination:False

Thirst:True

Blurred Vission:True

Dry mouth:True

Smelling breath:True

Shorness of breath: True

We obtained the result after submitting the form:

Warning! You could be diabetic. You need to make an appointment to see a doctor.

The result is accurate.

The system is programmed to imitate an expert human in a specific domain but still the expert is a human, which can make mistakes. However, even if the prediction of a real expert in medicine was sometimes wrong we would still trust them .The software can evaluate the risks of the patient having diabetes but still there are always exceptions in human world.

Likewise , at least in most cases we can rely on solutions provided by expert systems , but mistakes are possible and we should be aware of it.

In expert systems, knowledge is separated from its processing (the knowledge base and the inference engine are split up). A conventional program is a mixture of knowledge and the control structure to process this knowledge. This mixing leads to difficulties in understanding and reviewing the program code, as any change to the code affects both the knowledge and its processing. When an expert system shell is used, a knowledge engineer or an expert simply enters rules in the knowledge base. Each new rule adds some new knowledge and makes the expert system smarter.