Report: Expert system for car diagnosis

Members:

- Johan Aguirre
- Rafaela Ruiz
- Alejandro Amú.

Introduction

The Car Diagnostic Expert System is projected to assist users, especially young adults with little or no experience in automotive mechanics, in diagnosing ordinary problems with their vehicles. By adopting an expert system in the form of a chatbot, the system hopes to offer immediate, accessible, and practical solutions to starting problems, air conditioning malfunctioning, lighting issues, engine failure, and electrical faults in cars.

The report provides a description of the overall scope, design, implementation, testing, and future potential for the system.

Problem Statement and Objectives

Problem Statement:

Car owners often experience difficulties diagnosing and resolving vehicle issues due to a lack of technical knowledge. Traditional vehicle diagnostics can be costly, time-consuming, and may require specialized tools and services that are not always accessible. There is a need for a more accessible, digital solution to help car owners identify problems and implement basic solutions.

Objectives:

- Develop an expert system that can diagnose common automotive issues based on user-input symptoms.
- Provide easy-to-understand solutions and recommendations that do not require specialized equipment.
- Ensure that the system is accessible to users without technical expertise in automotive mechanics.
- Offer a chatbot interface that provides real-time diagnoses and advice.

Requirements Analysis

Functional Requirements.

Interaction Chatbot:

The system must function as an intuitive chatbot, where the user can enter symptoms observed on the vehicle.

The chatbot must be able to handle multiple symptoms at once to provide a more accurate diagnosis.

It should provide explanations of each symptom and possible causes, along with recommendations for actions the user can try.

Symptom-Based Diagnosis:

The system should include at least 20 diagnostic problems, 10 of which should be based on multiple conditions, increasing diagnostic accuracy.

The system must offer solutions that can be performed without specialized equipment and are safe for the user.

User Friendly Interface:

The interface should be designed for a young audience, be visually appealing and easy to use.

It should allow the user to select or enter symptoms without complexity.

Non-functional Requirements

- Accessibility and Availability:

The system must be accessible at any time to serve users at the time they are presented with the automotive problem.

The application must be easy to use and accessible to people with basic knowledge in the use of technology.

Scalability:

The system must be ready to receive and process queries efficiently as the number of users increases.

Maintainability:

The system should be easily upgradable to include new problems and solutions in the future, according to evolving problems and user needs.

For more information about the requirements, user interaction flow and project scope consult the requirements analysis document

Knowledge Acquisition and Representation

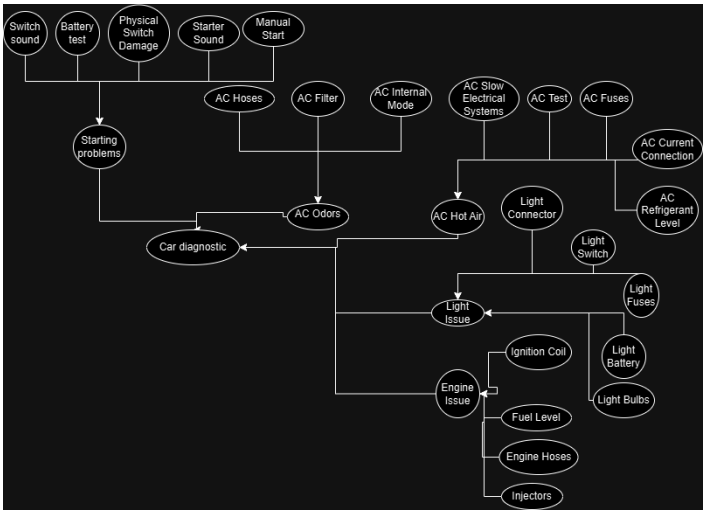
For the Knowledge Acquisition we did an interview with an experienced mechanic, and we consulted the user guide for common problems of multiple vehicles (the interviews can be consulted in the interview folder of the project), and we reach the most common problems presented in the vehicles that were categorized in 5:

- Starting problems
- Air conditioning operation problems (divided into two parts Bad odors and Hot air)
- Problems with the vehicle lights
- Engine problems
- Electrical problems

With the help of the expert, we reach also the question that needs to be asked to make a correct diagnosis and the probability of the solution being correct

For more detailed information about in the document Knowledge Acquisition

With all the information we realize the Bayesian network in the diagram that finishes being in the next way:



System Design

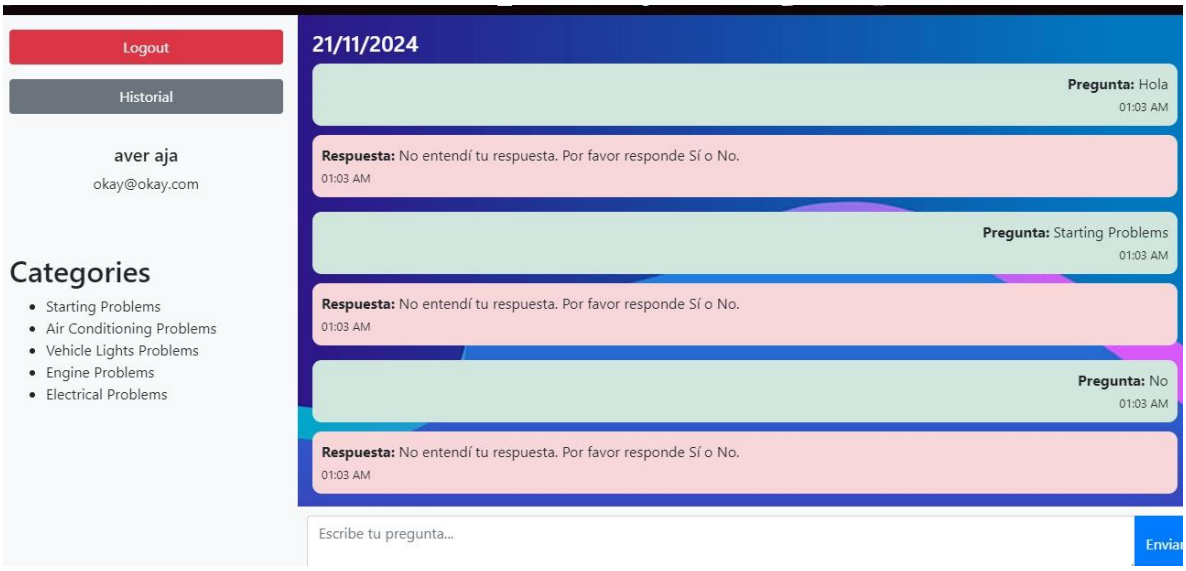
Regarding the system design, the following actions were carried out:

Regarding the graphic interface, one was made for validations and tests and another interface with which the final users of the application would interact, which looked as follows:

CLI interface:

```
Bienvenido al sistema experto de diagnóstico de vehículos
Seleccione el sistema a diagnosticar:
1. Problemas de arranque
2. Problemas de aire acondicionado
3. Problemas de luces
4. Problemas de motor
5. Problemas eléctricos
6. Salir
Ingrese el número de su opción: 5
¿Cual es el nivel de carga de la bateria? (Low/Charged): charged
¿Cual es el nivel de carga de la bateria? (damaged/Charged): damaged
Los problemas eléctricos causados posiblemente sean por el alternador dañado. Debe ser reemplazado.

¿Desea diagnosticar otro sistema? (si/no): no
Gracias por usar el sistema experto de diagnóstico de vehículos. ¡Hasta luego!
```

Web interface :



The database management is actually a series of json files that store the history of user conversations with the Bot and these files are stored in a SQLite database.

For the inference engine we used the probabilities given by the expert and the Bayesian network previously designed which looks as follows:

```
model = BayesianNetwork([
    ('Switch_Sound', 'Starting_Problems'),
    ('Battery_Test', 'Starting_Problems'),
    ('Physical_Switch_Damage', 'Starting_Problems'),
    ('Starter_Sound', 'Starting_Problems'),
    ('Manual_Start', 'Starting_Problems')
])

# 2. Definir las tablas de probabilidad condicional (CPTs)
# Probabilidad de cada causa (prior)
cpd_switch_sound = TabularCPD(
    variable='Switch_Sound', variable_card=2,
    values=[[0.25], [0.75]],
    state_names={'Switch_Sound': ['Yes', 'No']}
)

cpd_battery_test = TabularCPD(
    variable='Battery_Test', variable_card=2,
    values=[[0.5], [0.5]],
    state_names={'Battery_Test': ['Pass', 'Fail']}
)

cpd_physical_switch_damage = TabularCPD(
    variable='Physical_Switch_Damage', variable_card=2,
    values=[[0.1], [0.9]],
    state_names={'Physical_Switch_Damage': ['Yes', 'No']}
)

cpd_starter_sound = TabularCPD(
    variable='Starter_Sound', variable_card=2,
    values=[[0.1], [0.9]],
    state_names={'Starter_Sound': ['Normal', 'Weird']}
)

cpd_manual_start = TabularCPD(
    variable='Manual_Start', variable_card=2,
    values=[[0.05], [0.95]],
    state_names={'Manual_Start': ['Possible', 'Not_Possible']}
)
```

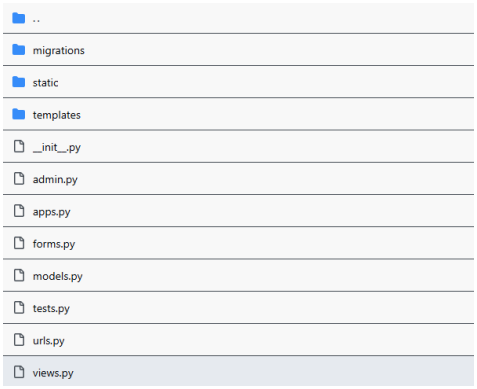
Implementation Details

For the implementation of the expert system, we made use of the experta and pgmpy libraries, additionally the framework used for the front end was Django.

Code fragment of the implementation of the expert with experta:

```
class DiagnosticoVehiculos(KnowledgeEngine):
    #  ----------------------------
    #  1. Problemas de Arranque
    #  ----------------------------
    @Rule(Match(Switch_Sound="yes"))
    def rule_car_not_starting(self):
        print("Debido a que el switch hace un ruido raro, es necesario realizar mantenimiento")


    @Rule(AND(Match(Battery_Test="fail"), Match(Switch_Sound="no")))
    def rule_weird_switch_sound(self):
        print("Debido a que el test de fallo, es posible que se deba cambiar la bateria")

    @Rule(AND(Match(Physical_Switch_Damage="yes"), Match(Switch_Sound="no"), Match(Battery_Test="pass")))
    def rule_dry_sound_failed_battery(self):
        print("Es posible que el problema sea el switch o la llave debido al mal estado, necesitan ser ajustadas")

    @Rule(OR(Match(Starter_Sound="weird"), Match(Manual_Start="not_possible")))
    def rule_dry_sound_failed_battery(self):
        print("Se debe llamar un experto para realizar el reemplazo del sistema de arranque.")


    # ----------------------------
    # 2. Problemas de Aire Acondicionado
    # ----------------------------

    # Malos Olores
    @Rule(Match(AC_Odors="yes"))
    def rule_ac_bad_smell(self):
        print("El aire acondicionado huele mal. Revisa si el filtro está obstruido o las mangueras están dañadas.")

    @Rule(AND(Match(AC_Odors="yes"), Match(AC_Filter="obstructed")))
    def rule_ac_dirty_filter(self):
```

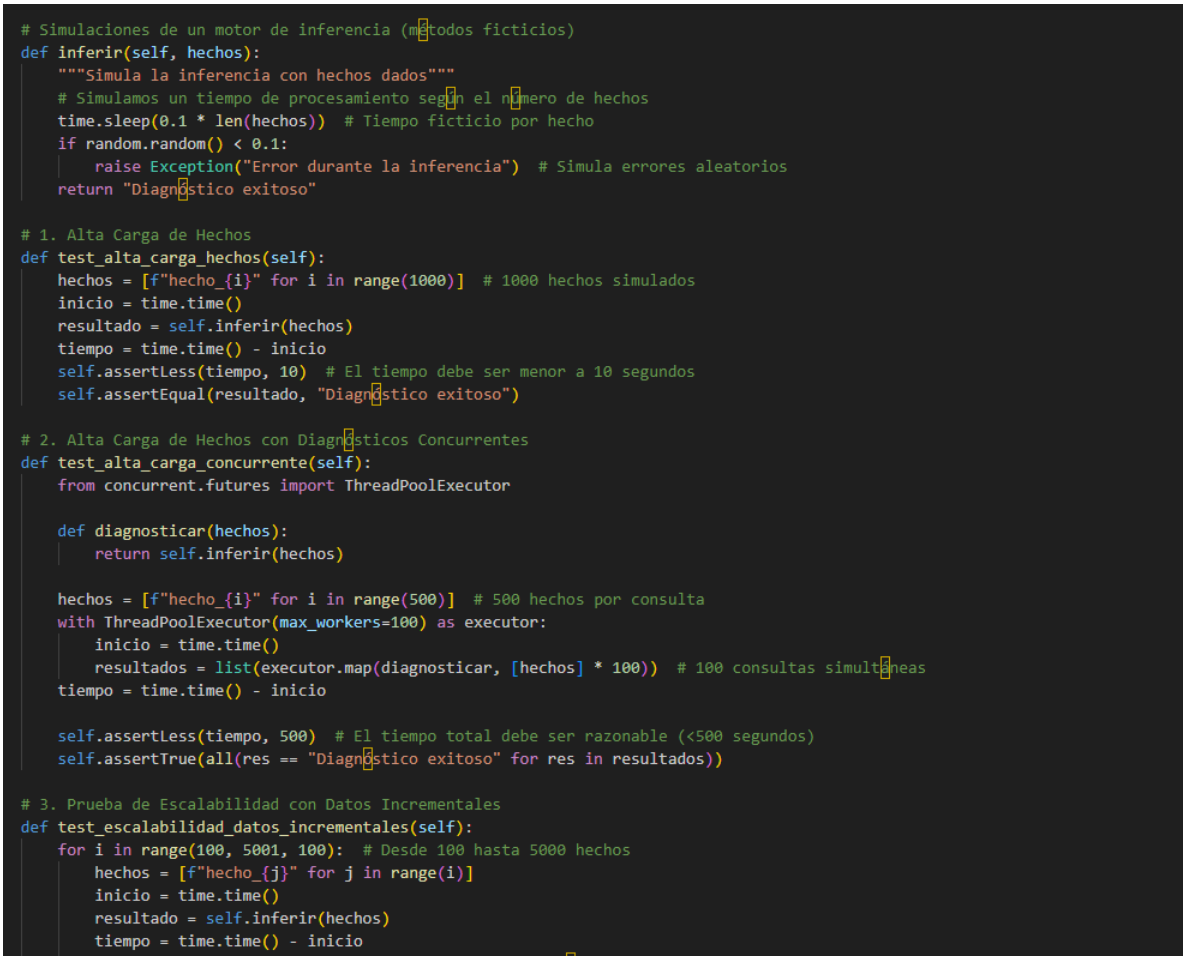
Use of the Django framework:



Testing and Validation

For validation and testing we first made a detailed test plan on how to handle both unit tests and tests with real users using prototypes in this case two prototypes were presented, the one that made use of the CLI and the web interface.

First the unit tests were performed, which range from the load capacity of the application without errors, the average response time, etc. The results in the undeployed version were satisfactory, but in the deployed version the results were altered due to the extra time needed for the assembled application to perform queries between the back and the front.



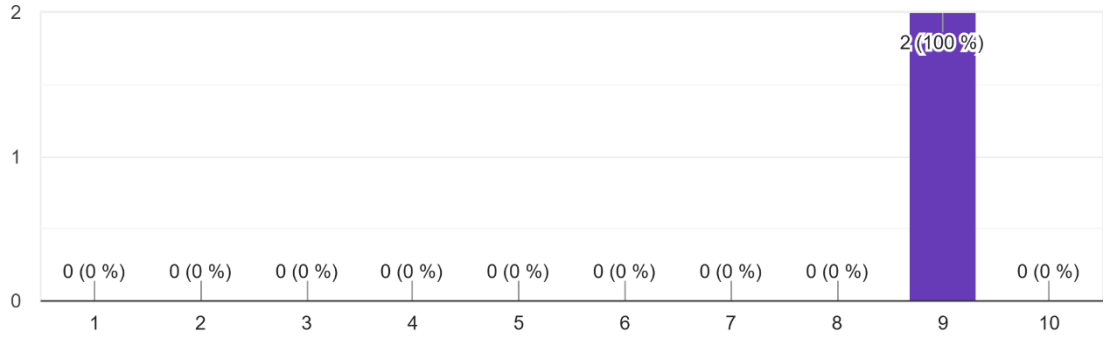
```python
# Simulaciones de un motor de inferencia (métodos ficticios)
def inferir(self, hechos):
    """Simula la inferencia con hechos dados"""
    # Simulamos un tiempo de procesamiento según el número de hechos
    time.sleep(0.1 * len(hechos))  # Tiempo ficticio por hecho
    if random.random() < 0.1:
        raise Exception("Error durante la inferencia")  # Simula errores aleatorios
    return "Diagnóstico exitoso"

# 1. Alta Carga de Hechos
def test_alta_carga_hechos(self):
    hechos = [f"hecho_{i}" for i in range(1000)]  # 1000 hechos simulados
    inicio = time.time()
    resultado = self.inferir(hechos)
    tiempo = time.time() - inicio
    self.assertLess(tiempo, 10)  # El tiempo debe ser menor a 10 segundos
    self.assertEqual(resultado, "Diagnóstico exitoso")

# 2. Alta Carga de Hechos con Diagnósticos Concurrentes
def test_alta_carga_concurrente(self):
    from concurrent.futures import ThreadPoolExecutor

    def diagnosticar(hechos):
        return self.inferir(hechos)

    hechos = [f"hecho_{i}" for i in range(500)]  # 500 hechos por consulta
    with ThreadPoolExecutor(max_workers=100) as executor:
        inicio = time.time()
        resultados = list(executor.map(diagnosticar, [hechos] * 100))  # 100 consultas simultáneas
    tiempo = time.time() - inicio

    self.assertLess(tiempo, 500)  # El tiempo total debe ser razonable (<500 segundos)
    self.assertTrue(all(res == "Diagnóstico exitoso" for res in resultados))

# 3. Prueba de Escalabilidad con Datos Incrementales
def test_escalabilidad_datos_incrementales(self):
    for i in range(100, 5001, 100):  # Desde 100 hasta 5000 hechos
        hechos = [f"hecho_{j}" for j in range(i)]
        inicio = time.time()
        resultado = self.inferir(hechos)
        tiempo = time.time() - inicio
```

On the other hand, tests were also performed with users using both graphical interfaces and validations were performed with the expert that resulted in improvements in the probabilities given.

Results of the post-test user surveys (https://forms.gle/y4SCMk4QXmtgPXnu7):

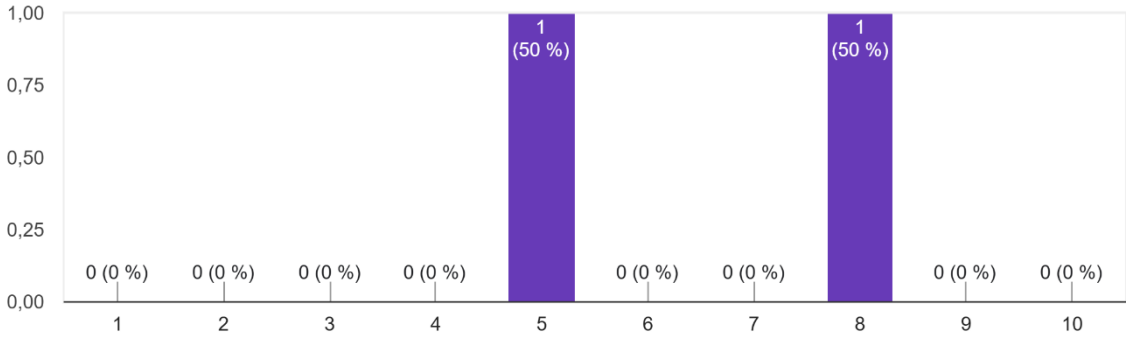
En una escala del 1 al 10 por favor puntué su experiencia con la aplicación

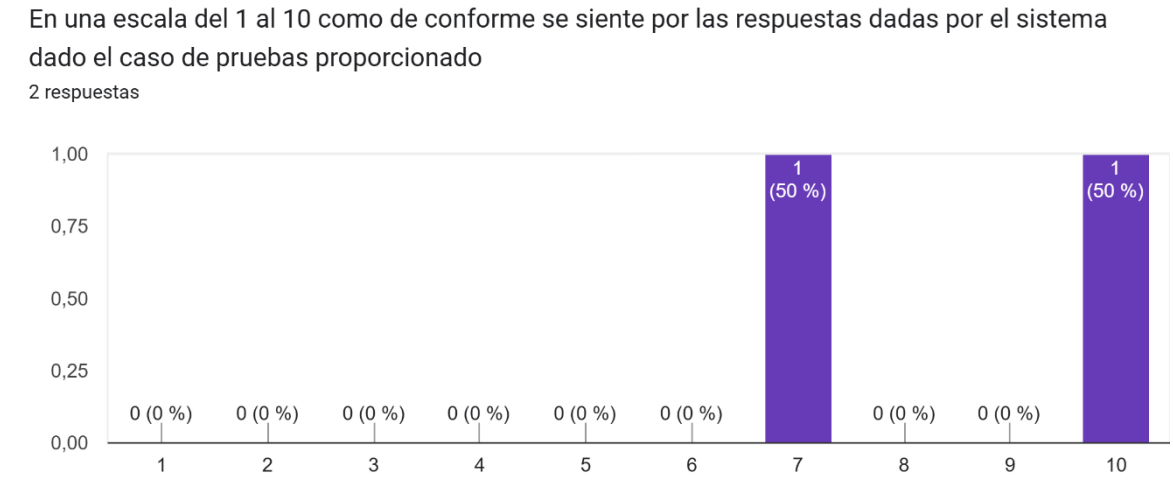
2 respuestas

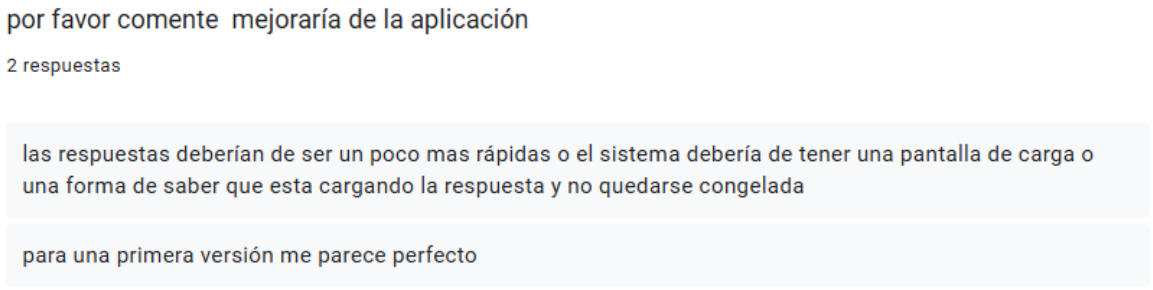


En una escala de 1 a 10 por favor puntué la velocidad del sistema a la hora de hacer la consulta de los casos de prueba

2 respuestas







For details on the tests themselves see the Test plan design document, for more in-depth user and expert reviews see the interviews folder.

Deployment

For the deployment, the application was mounted on a free hosting system such as vercel, however we had problems when mounting the database with the system because we did not find a known hosting that allowed us to mount the database correctly (Rip elefhant sql).

Conclusion and Future Work

The Car Diagnostic Expert System is an innovative solution that leverages expert system technologies to provide accessible and practical vehicle diagnostics. The system meets the primary objectives of assisting users in diagnosing common car problems and offering solutions in a user-friendly manner.

Future Work:

- **Expansion of Knowledge Base:** Add more car problems and solutions to enhance the diagnostic capabilities.

- **Multilingual Support:** Expand the system to support multiple languages to reach a broader audience.
- **Mobile Application:** Develop a mobile version of the system for greater accessibility.

References

https://www.team-bhp.com/forum/technical-stuff/121153-flow-charts-troubleshooting-car-problems.html

https://www.onallcylinders.com/2016/12/14/infographic-guide-to-diagnosing-common-starting-problems/