Universidad Icesi

Cristian Camilo Rivadeneira
A00354996
Juan Andrés Orozco Núñez
A00355202
Josué Rodríguez Pineda
A00359703

Facultad de Ingeniería de Sistemas
Santiago de Cali
2020

Identificación del problema

Durante los últimos años, la compra de vehículos en el continente Europeo se ha convertido en algo habitual fruto del gran progreso económico, y es que por ejemplo Liechtenstein, principado de habla alemán ubicado entre Austria y Suiza, es el segundo país después de Estados Unidos en tener más vehículos por persona. Este fenómeno ha conllevado que las diferentes empresas automotrices a nivel global encuentren un mercado atractivo en estas regiones y, por ende, hayan introducido sus modelos de vehículos.

Sin embargo, muchos de los reportes provenientes de encuestas recientes indican que los diferentes consumidores han podido identificar diversas dificultades en torno a la anterior situación. En primer lugar, la existencia de una gran oferta automotora provoca una gran incertidumbre a la hora de escoger un vehículo adecuado, y en segundo lugar, la poca información sobre si los modelos ofrecidos son aparatos que se ofrecen en condiciones óptimas para el consumidor final.

Identificación de necesidades

- Se requiere gestionar la información de los diferentes vehículos actualmente disponibles en el mercado europeo.
- Se requiere poder clasificar dicha información en categorías que permitan agrupar las características de los vehículos.
- Se requiere poder generar consolidados sobre la clasificación de la información de los vehículos.

- Se requiere poder clasificar la información de los diferentes vehículos en apartados por categorías que permitan decidir si un vehículo es una buena o mala opción de compra.
- Se requiere poder visualizar dicha clasificación de tal modo que pueda ser comprendida por los diferentes usuarios que vayan adquirir el vehículo.

Identificación del problema

Se requiere diseñar un aplicativo que permita analizar la información de diferentes vehículos, tales como precio de venta, precio de mantenimiento, número de puertas, entre otros, para poder clasificarlos y decidir si el automóvil es una opción viable de compra o, si, por el contrario, la compra del automóvil es una pésima opción y generar reportes sobre las mejores marcas de vehículos ofrecidos actualmente en el mercado europeo.

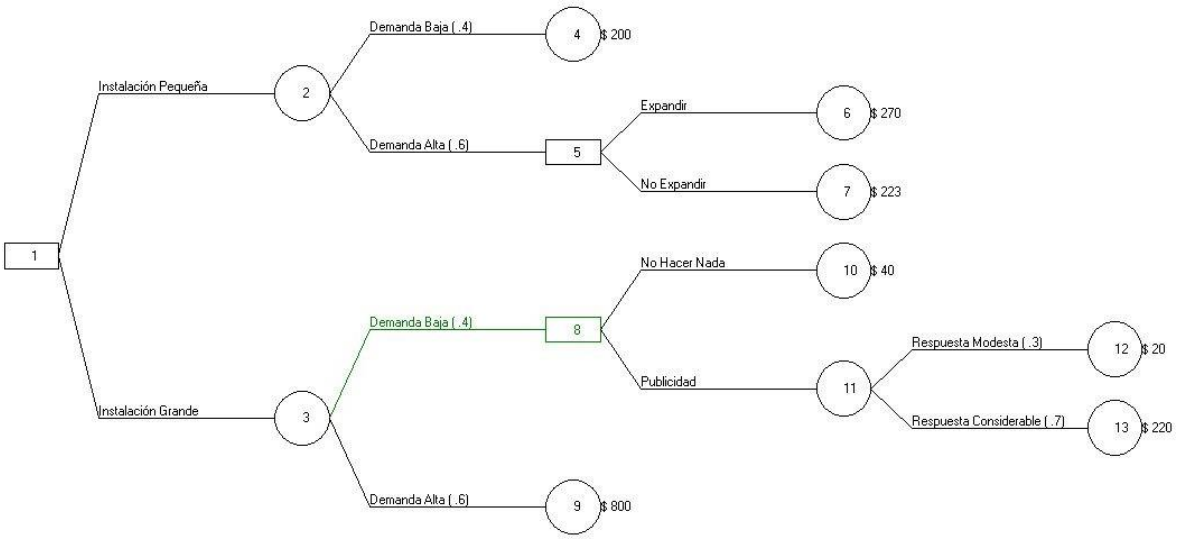
Recopilación de la información necesaria

Marco teórico.

El análisis exploratorio tiene como objetivo identificar el modelo teórico más adecuado para representar la población de la cual proceden los datos muestrales. Dicho análisis se basa en gráficos y estadísticos que permiten explorar la distribución identificando características tales como: valores atípicos o outliers, saltos o discontinuidades, concentraciones de valores, forma de la distribución, etc. Por otra parte, este análisis se puede realizar sobre todos los casos conjuntamente o de forma separada por grupos. En este último caso los gráficos y estadísticos permiten identificar si los datos proceden de una o varias poblaciones, considerando la variable que determina los grupos como factor diferenciador de las poblaciones. También permite comprobar, mediante técnicas gráficas y contrastes no

paramétricos, si los datos han sido extraídos de una población con distribución aproximadamente normal. (Universidad de Barcelona, 2008)

Árboles de Decisión. Técnica que permite analizar decisiones secuenciales basada en el uso de resultados y probabilidades asociadas. Los árboles de decisión se pueden usar para generar sistemas expertos, búsquedas binarias y árboles de juegos. Los árboles de decisión se utilizan en cualquier proceso que implique toma de decisiones, Los árboles de decisión generalmente son binarios, es decir que cuentan con dos opciones, aunque esto no significa que no puedan existir árboles de tres o más opciones (Aaron, 2010)



Índice Gini:

Se utiliza para atributos con valores continuos (precio de una casa). Esta función de coste mide el "grado de impureza" de los nodos, es decir, cuán desordenados o mezclados quedan los nodos una vez divididos. Deberíamos minimizar ese GINI Índex. (Na8, 2018)

Ganancia y Entropía:

El algoritmo ID3 decide cuál atributo es mejor por medio de una propiedad estadística llamada ganancia de información. Esta ganancia mide cuán bien un atributo dado separa los ejemplo del conjunto de pruebas en clases específicas. La que contiene la mayor cantidad de información (Siendo la información la más útil para la clasificación). Para definir esta ganancia primero debemos obtener información acerca de la teoría de entropía. La entropía mide la cantidad de información en un atributo.

Dada una colección S de c se tiene

$$Entropia(S) = \ S - p(I)\ log_{\,2}\, p(i)$$

Donde p(I) es la proporción de S que pertenece a la clase I. S está sobre c. Nótese que S no es un atributo sino el conjunto de datos de ejemplos.

Ejemplo

Suponga que S es un conjunto de 14 ejemplos los cuales uno de ellos es la velocidad del viento.Los valores de viento pueden ser *Weak* o *String.* La clasificación de estos 14 ejemplos son 9 SI y 5 NO. Entonces tenemos 8 ocurrencia del Viento iguales a *Weak* y 6 ocurrencias de Viento = *Strong*, tenemos 6 ejemplo de SI y 2 de No.Pra Viento = *Strong, 3* son SI y 3 son NO. Entonces

$Ganancia(S, V\,iento) = Entropia(S) - (8/14) * Entropia(Sweak\ ) - (6/14) * Entropia(Sstrong\ )$

= 0.940 - (8/14)*0.811 - (6/14)*1

= 0.48

$Entropia(S_{weak}\ ) = - 6/8 * log_2(6/8) - (2/8) * log_2(2/8) = 0.811$

$Entropia(S_{strong}\ ) = - 3/6 * log_2(3/6) - (3/6) * log_2(3/6) = 1$

Estado del Arte

Para el desarrollo y solución del problema manejaremos una estrategia de machine learning la cual usa los árboles de decisión como herramienta para clasificar un conjunto de datos, y decidir diferentes conclusiones con base a estos. En la etapa de diseño se ha decidido que será implementado usando una construcción propia, es decir, sin utilizar librerías externas ya programadas, en el lenguaje de programación C#. Con lo cual se requiere una investigación formal en diversas maneras de abordar este problema usando las herramientas disponibles.

A continuación se muestra una tabla con las posibles soluciones para construir un ID3:

Opciones	Ventajas	Desventajas
Grafos	● Manejo de un alto volumen de datos. ● Retornan soluciones eficaces a problemas complejos.	● Es más difícil de programar. ● Complejidad temporal alta.
Árboles Binarios	● Manejo de un alto volumen de datos ● Es más fácil de programar. ● Complejidad temporal baja.	● No dependen de una estructura discreta por lo que las soluciones que ofrecen se remiten a problemas relativamente sencillos.

Tabla 1*. Comparación de estructuras de datos que pueden ser usadas para construir un ID3*

Como podemos observar en la ***Tabla 1*** es mucho más conveniente utilizar estructuras de árboles, en lugar de grafos, para abordar la construcción del algoritmo de árboles de decisión, además de que teóricamente tienen un funcionamiento similar, tal como sus nombres lo indican. El hecho de que sean utilizados en problemas sencillos, generalmente depende del contexto del problema, pero no impide que se puedan llevar a niveles algorítmicos más complejos.

El algoritmo ID3: Para construir un algoritmo que represente a un árbol de decisión hay que tener en cuenta que ID3 es un algoritmo no incremental, lo que significa que deriva sus clases de un conjunto fijo de instancias de entrenamiento. Un algoritmo incremental revisa la definición del concepto actual, si es necesario, con una nueva muestra. Las clases creadas por ID3 son inductivas, es decir, dado un pequeño conjunto de instancias de entrenamiento, se espera que las clases específicas creadas por ID3 funcionen para todas las instancias futuras. Usando la definición anterior, se inició el proceso de programación usando las nociones básicas de un árbol binario. Los elementos como raíz, rama, hoja son fundamentales ya que con ellos podemos estructurar el funcionamiento básico de nuestro algoritmo. El criterio de funcionamiento en la construcción de las diferentes opciones y salidas del árbol de decisión vienen determinadas por un elemento conocido como atributo. Este atributo es el punto clave que el algoritmo utiliza para definir cuál será la raíz y cuáles serán las ramas del árbol. Una vez obtenido estos datos el algoritmo encuentra las hojas, las cuales son las clases de salida que permiten decidir qué función tomar con base a un ejemplo diferente que requiera ser comparado.

Clase/Elemento del árbol	Funciones
Root	● Administrar el atributo raíz obtenido a partir de la ganancia. ● A partir del valor del atributo raíz elegir qué atributos del dataset son útiles para obtener una clasificación adecuada. ● Evitar usar clases de resultado como atributos. ● Eliminar los ejemplos del dataset que no resultan relevantes para tomar una decisión
Node	● Sirve como contenedor temporal de todos los atributos que el nodo Root quiera almacenar para luego ser incorporados al árbol.
Attribute	● Elemento que representa un atributo del dataset en el árbol. Permite tomar decisiones sobre los nodos mediante cálculos realizados desde Root.

Tabla 2. *Comparación de estructuras de datos que pueden ser usadas para construir un ID3*

La ***Tabla 2*** especifica las diferentes funciones que cada elemento del árbol debe realizar para funcionar como un árbol de decisión. Cada elemento representa una clase en el lenguaje de programación de C# y sus funciones son métodos o instrucciones que cumplen con las tareas especificadas. Una vez implementadas como algoritmos, se integran en un modelo que siga los paradigmas de programación orientada a objetos para que puedan recibir nuestro set de entrenamiento y realizar la respectiva clasificación.

Búsqueda de soluciones creativas.

Ideas para la creación del aplicativo

- Crear una aplicación que permita el ingreso de un grupo de registros de un dataset y permita clasificarlos, mostrando las salidas correspondientes en una tabla.
- Utilizar la herramienta de control DataGridView para la visualización de los datos y sus respectivos reportes.

- Utilizar las librerías para árboles de decisión en el lenguaje Python y traducirlas posteriormente a C#.
- Crear una aplicación que se conecte a algún servidor externo (Azure o AWS) donde se procesa la información y retorna las clasificaciones correspondientes.
- Indagar repositorios en Github con alguna implementación realizada que nos pueda servir como guía para realizar la nuestra.
- Usar librerías gráficas para la creación de una visualización con forma de árbol que se adapte a los resultados de nuestro proyecto.

Ideas para la clasificación de los registros

- Para cada marca de vehículo se busca manualmente en internet sus características y de acuerdo al feedback de los usuarios se clasifica el vehículo.
- Se busca en alguna base de datos donde solo esté la clasificación y se retorna posteriormente.
- Para cada registro de un determinado grupo de autos se identifican las características más relevantes que puedan definir si el vehículo es bueno o no y con base a ellas se estandarizan los requisitos que debe cumplir un grupo de automóviles al ser clasificados.
- Obtenemos la clasificación de los vehículos haciendo uso de nuestro algoritmo de ID3.

Transición de la formulación de ideas a los diseños preliminares.

Tomando en cuenta los requerimientos solicitados para el proyecto, se ha decidido analizar sobre las siguientes alternativas.

Para el Aplicativo:

Alternativa 1: Ingreso del dataset

Dado que el objetivo de este proyecto es realizar un análisis con un reporte para las variables presentes es más que factible y necesario que el dataset sea leído y mostrado en una tabla en el aplicativo.

Alternativa 2: DataGridView

Las herramientas para visualización de datos que ofrece C# permiten construir código de manera rápida, limpia y eficiente. DataGridView es un elemento de control de Windows Forms que ofrece la posibilidad de mostrar una gran cantidad de datos sin la necesidad de un arduo consumo de recursos, como es esperado, ya que los training set para los ID3 superaran la cantidad mínima de 1000 ejemplos.

Alternativa 3: Python

A pesar de que Python es un lenguaje robusto en cuanto a Machine Learning se refiere, depende de sus librerías pre-programadas, las cuales nos fueron estrictamente prohibidas para esta etapa de desarrollo. Implementar dichas librerías requeriría mucho más trabajo el cual, dado al tiempo que tenemos, no se considera viable.

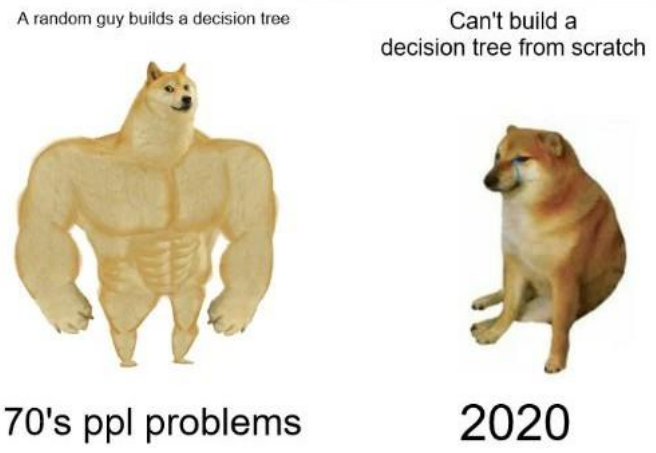
Alternativa 4: Azure

Aunque Azure sea un gran administrador de bases de datos e IA, no es útil para los objetivos que buscamos en esta parte del desarrollo del proyecto.

Alternativa 5: Github

Los repositorios de Github albergan una gran cantidad de implementaciones ya realizadas de diferentes proyectos. Al indagar en el tópico de ID3 se encuentran muchos repositorios que ya tienen o usan la funcionalidad de ID3. Sin embargo, muchos de ellos no implementan la estructura que nosotros buscamos la cual se asemeja a los árboles o usan librerías externas.

Después de una larga búsqueda encontramos un modelo basado en los apuntes de Andrew Colin y el Dr. Dobbs Journal el cual sigue los lineamientos que planteamos para la construcción del algoritmo. En este punto decidimos probar el código y depurarlo para iniciar nuestra propia implementación.



Alternativa 6: Graphics

Librerías integradas a C# cómo Graphiz o TreeView permiten el modelamiento de árboles binarios o n-arios por lo que pueden ser útiles para nuestro proyecto.

Para la clasificación de los registros:

Alternativa 1: Feedback

Si nuestro objetivo es usar un ID3 para construir un árbol de decisión que permita elegir que tan buena o mala opción sea un vehículo, no es para nada productivo ni confiable buscar feedback de personas al azar para decidir sobre un dataset que tiene una gran cantidad de datos.

Alternativa 2: Búsqueda

El objetivo de este proyecto es especifico al decir que nuestro objetivo es generar las rutas de clasificación para decidir sobre los automóviles. En el dataset ya viene dada una clasificación correspondiente, pero es gracias a ella que podemos obtener el árbol de decisión. Por lo tanto, será una obtener esta clasificación solo será de ayuda en la construcción del ID3.

Alternativa 3: Clasificación manual.

Una clasificación manual generara resultados con alto sesgo, no es una opción recomendada.

Alternativa 4: ID3

Ya que el objetivo de nuestro ID3 es que nos permita decidir sobre qué tan buena o mala opción de compra es un vehículo basado en sus características, optamos por usar los resultados de este algoritmo para realizar las conclusiones necesarias y que aporten a la solución del problema planteado.

Evaluación y selección de la mejor solución.

Criterios:

Criterio 1: Presentación y visualización de los reportes.

Ya que la solución debe permitir ver y filtrar por categorías el dataset, antes de la clasificación, es necesario que esta información se presente de manera que el usuario pueda visualizarla de manera sencilla e intuitiva en el programa.

1. La información se encuentra dispersa y es imposible filtrarla para obtener reportes sobre los datos presentes en el dataset.

2. Aunque es posible filtrar la información, este proceso no es del todo eficiente, siendo incompleto.

3. La información se filtra de manera correcta, cubriendo todas las categorías del dataset y presentando los resultados requeridos por el usuario.

Criterio 2: Carga y "entrenamiento" de los datos en un ID3 .

Los elementos del dataset deben ser divididos en dos subconjuntos, uno de prueba y otro de entrenamiento. El subconjunto de entrenamiento debe ser cargado en un ID3 para obtener las posibles variantes de decisión generadas por el algoritmo. Estas variantes deben permitir clasificar futuras entradas.

1. El ID3 no carga correctamente los datos y arroja una solución obsoleta que no cumple con los criterios de un buen árbol de decisión .

2. El ID3 carga el dataset, pero la salida muestra resultados incompletos al igual que pérdidas de atributos que son necesarios para obtener un buen árbol de decisión.

3. El ID3 carga el dataset, muestra una salida que agrupa todos los atributos importantes y permite comparar las salidas con futuras entradas para obtener una decisión.

Criterio 3: Visualización de la estructura del ID3 en un componente gráfico.

Una vez se ha cargado y entrenado el dataset, este debe ser mostrado en una interfaz que permita tener un acercamiento gráfico a los resultados obtenidos por el árbol, de esta manera el usuario podrá utilizarlo más fácilmente.

1. El ID3 no se muestra en una interfaz gráfica o la visualización no es amigable y poco intuitiva para el usuario
.
2. El ID3 se muestra en una interfaz gráfica que es amigable y que permite una interacción amigable e intuitiva con el usuario que vaya a operar el resultado obtenido por el algoritmo del árbol de decisión.

Alternativa (Aplicativo)	**Criterio**	**Puntaje según el criterio**
Alternativa 1	Criterio 1	3
Alternativa 2	No se adecua a ningún criterio.	Se descarta
Alternativa 3	No se adecua a ningún criterio.	Se descarta
Alternativa 4	Criterio 2	3

Alternativa 5	Criterio 3	3

Evaluación:

De las alternativas explicadas elegimos las siguientes:

Para el aplicativo:

- Alternativa 1: Las herramientas de DataGridView son bastante eficientes en para tareas de visualización de datos. Además, integra elementos que son útiles para generar reportes y filtros por categorías que son sencillo y fáciles de usar para un usuario final.

- Alternativa 4: El código encontrado en los repositorios de Github, después de su procesamiento, permitió la construcción de un ID3 acorde a todos los requerimientos necesarios para su buen funcionamiento. Además, este algoritmo será util para cualquier dataset que sea ingresado.

- Alternativa 5: Usamos la librería TreeView para generar la respectiva visualización gráfica del árbol, sus métodos se acoplaron perfectamente al modelo de árbol binario que planteamos para la base de nuestro ID3. También, resultó sencilla su construcción gracias a la facilidad de acople con archivos de texto.

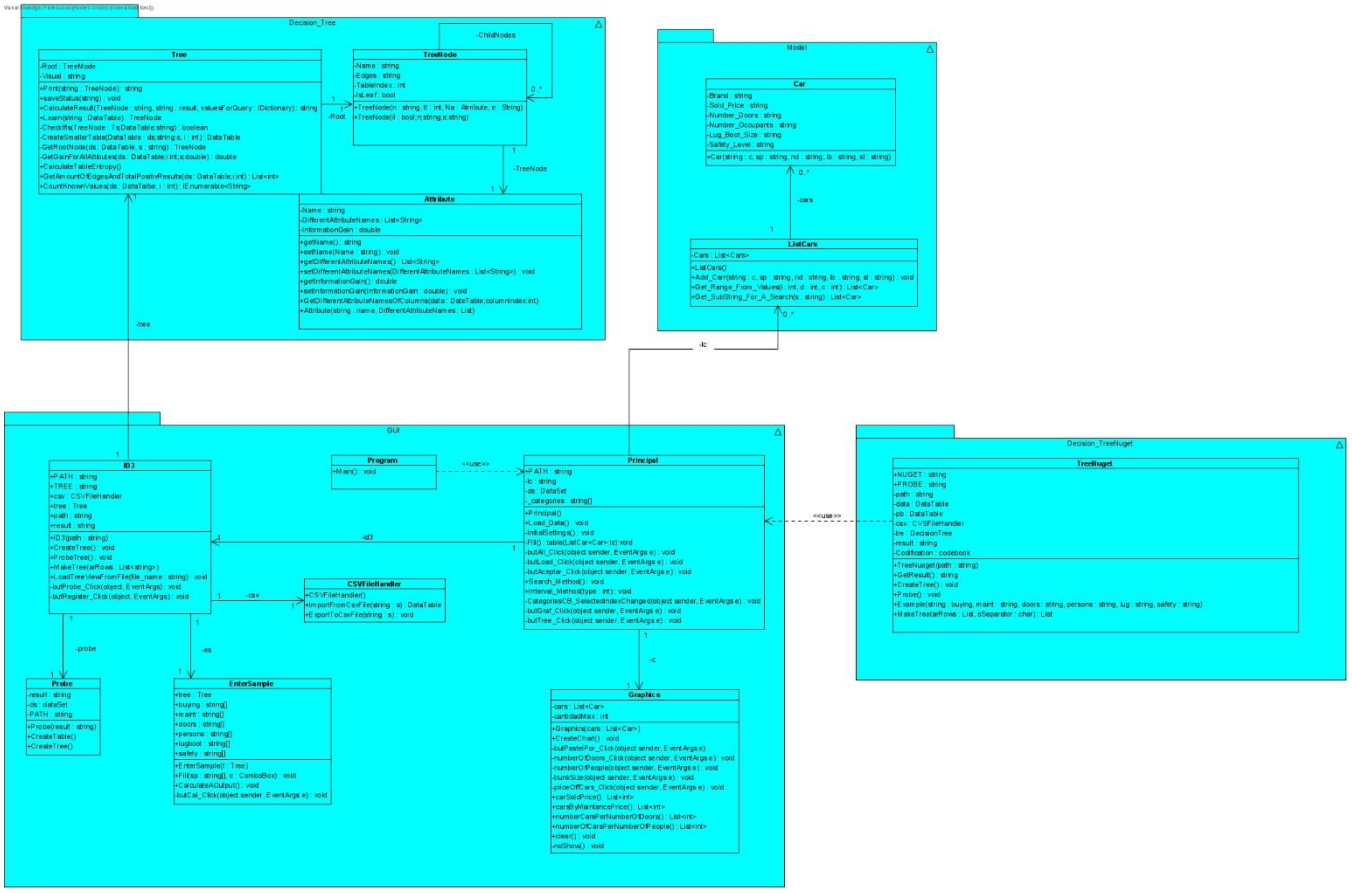
Para la clasificación de los registros:

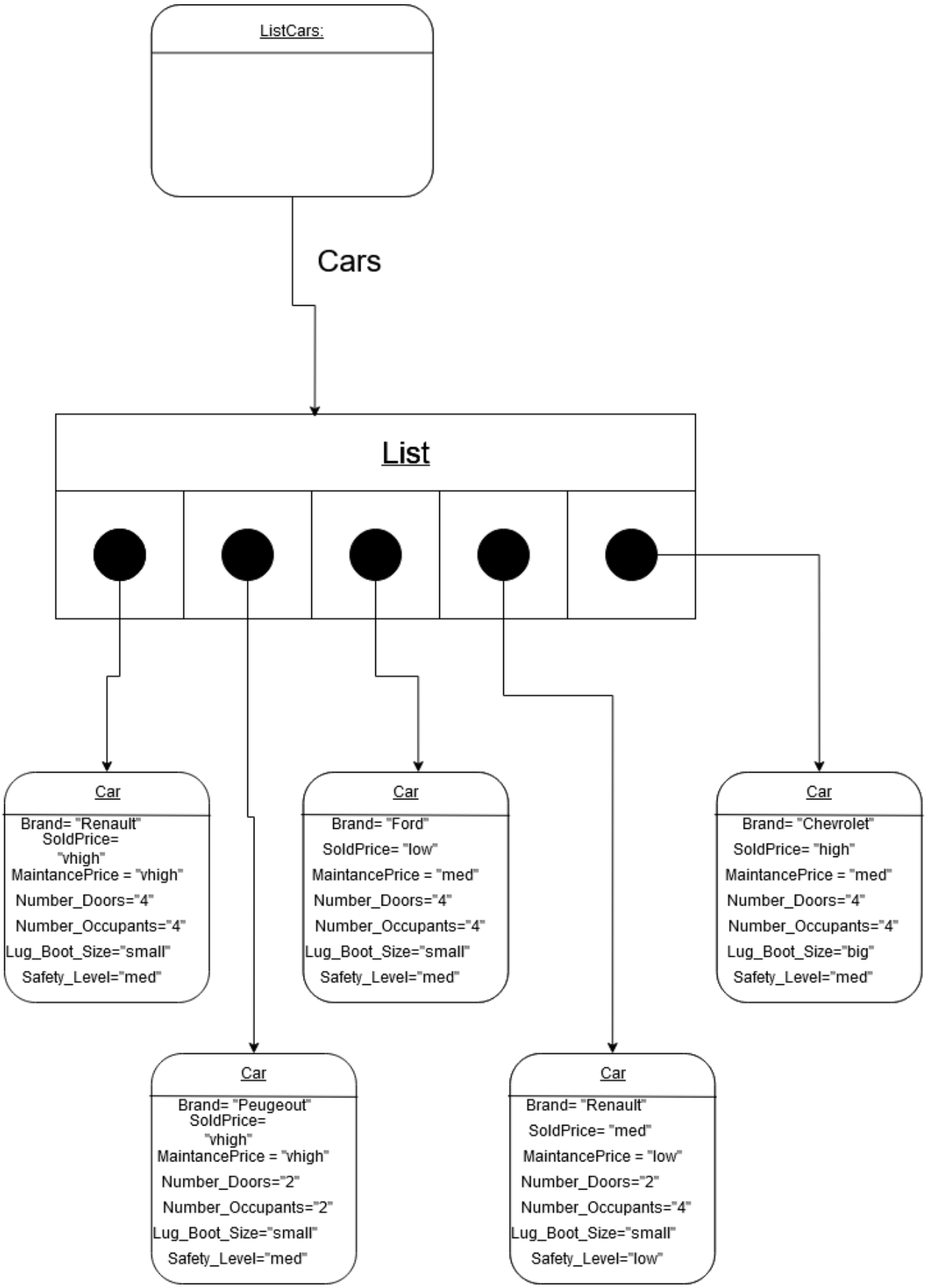
- Alternativa 2: Gracias al ID3 construido, es posible obtener resultados para clasificar las características de los vehículos, y poder decidir qué tan buena o mala opción es un vehículo de determinadas características.

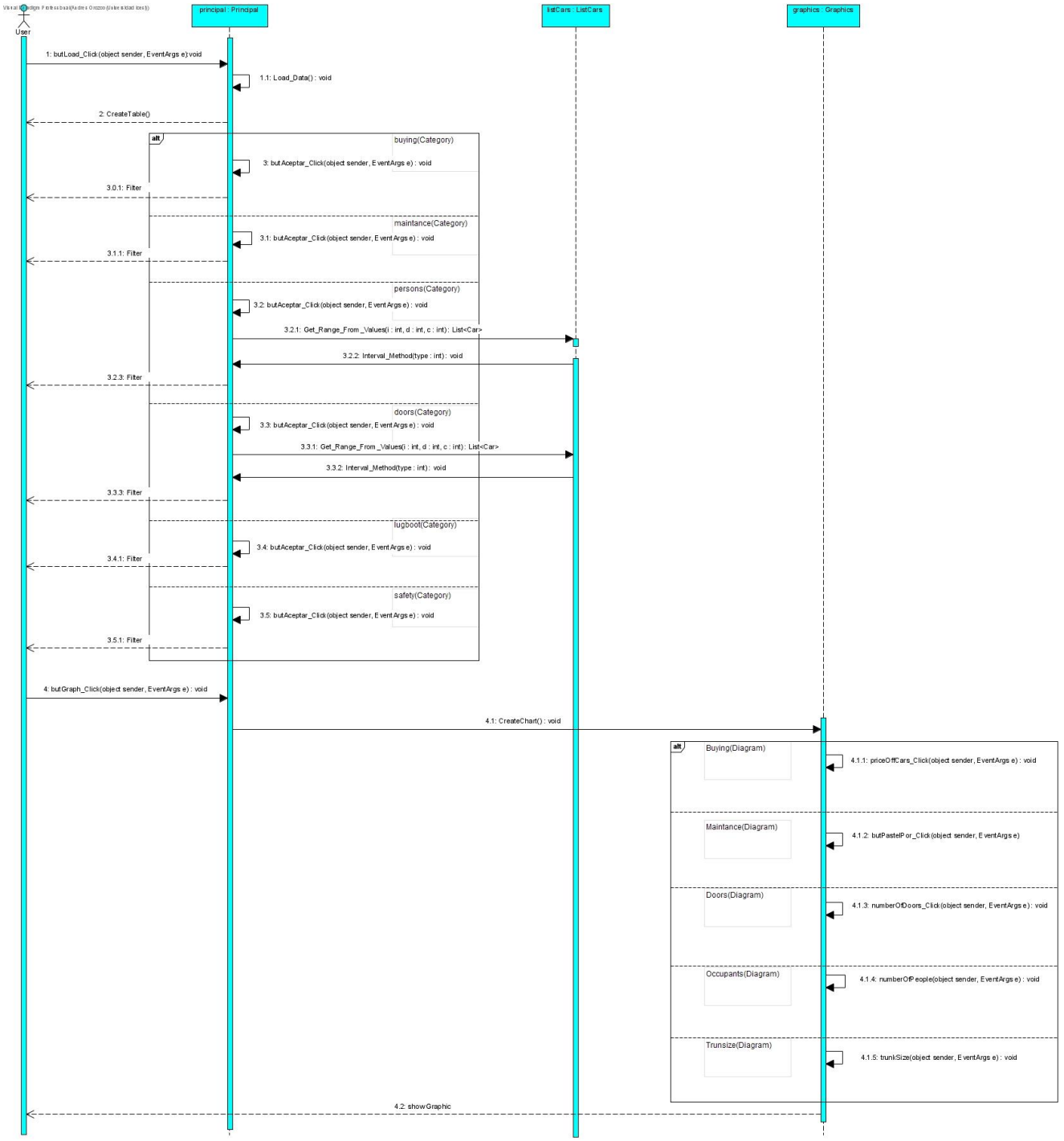
Síntesis Reflexiva

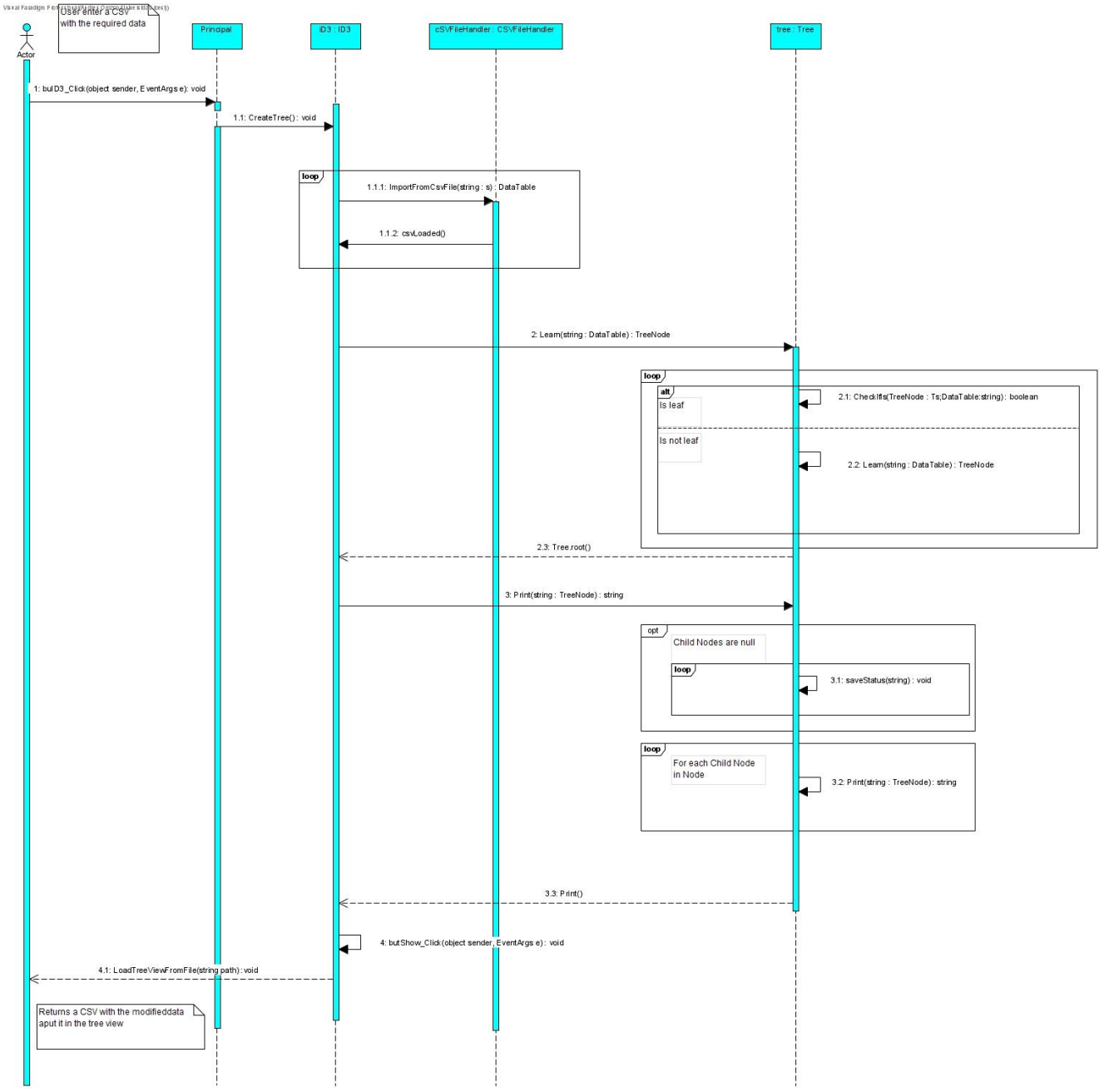
En esta parte de nuestro trayecto como estudiantes de ingeniería hemos notado que la relevancia del método de la ingeniería se ha incrementado. Gracias al diseño realizado en este proceso es mas sencillo afrontar el código para escribirlo de una manera mas sistemática y ordenada. Debido a los limites del proyecto las soluciones fueron oportunas y encaminadas a dar una solución acertada a lo que nos planteamos como objetivos. Un elemento importante, que queremos destacar, es que los trabajos de investigación realizados para este proyecto nos ayudaron a expandir nuestro panorama para la solución de problemas. También hemos afianzado la comunicación y el trabajo en equipo, creando roles para una mejor distribución de tareas. En este punto creemos que lo que hemos logrado es una solución bastante efectiva y que a medida que avancemos adquiriendo nuevos conocimientos lograremos mejorar las propuestas que realicemos para diversos problemas, que incluyan o no, nuestro campo de acción como ingenieros de sistemas.

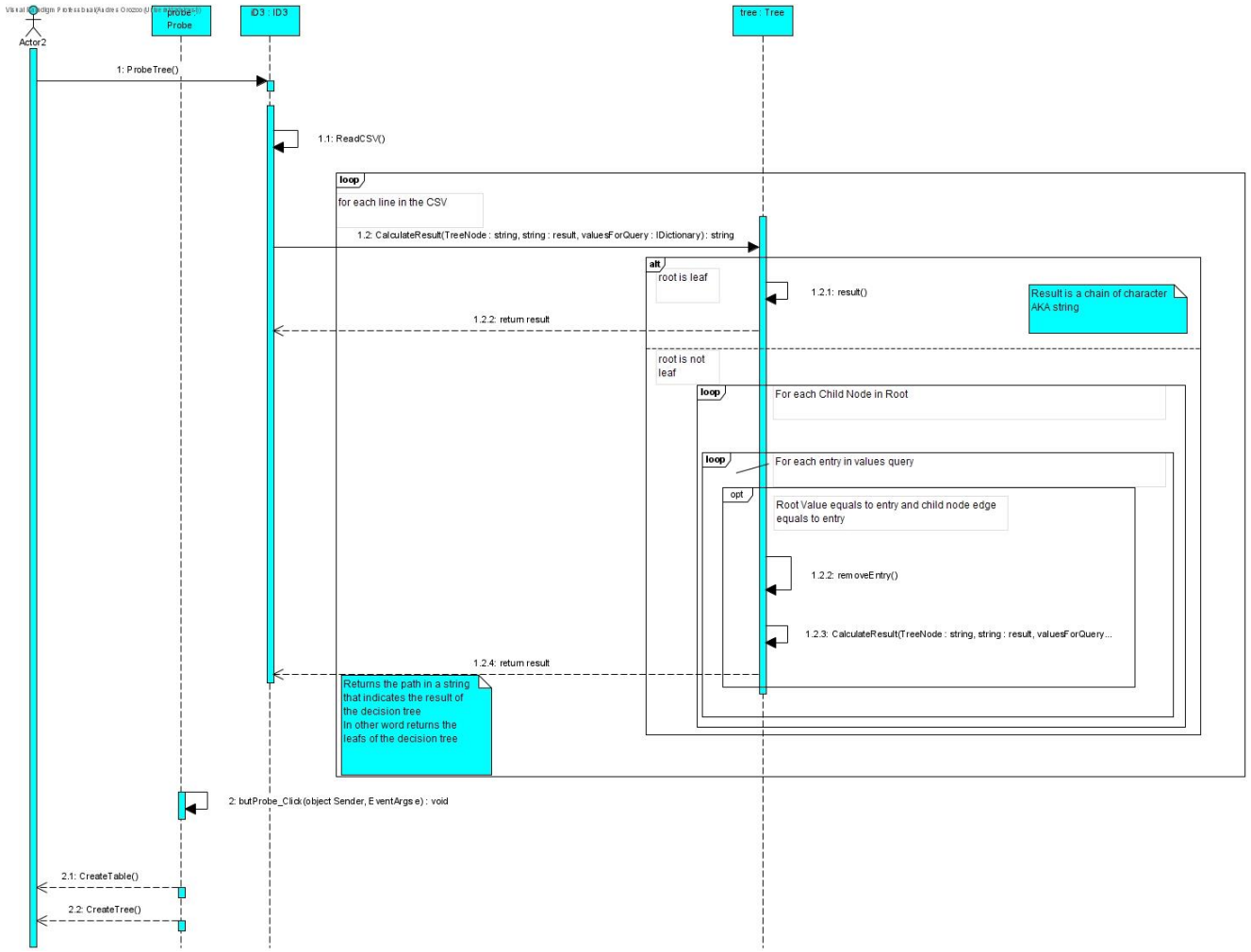
Diagramas

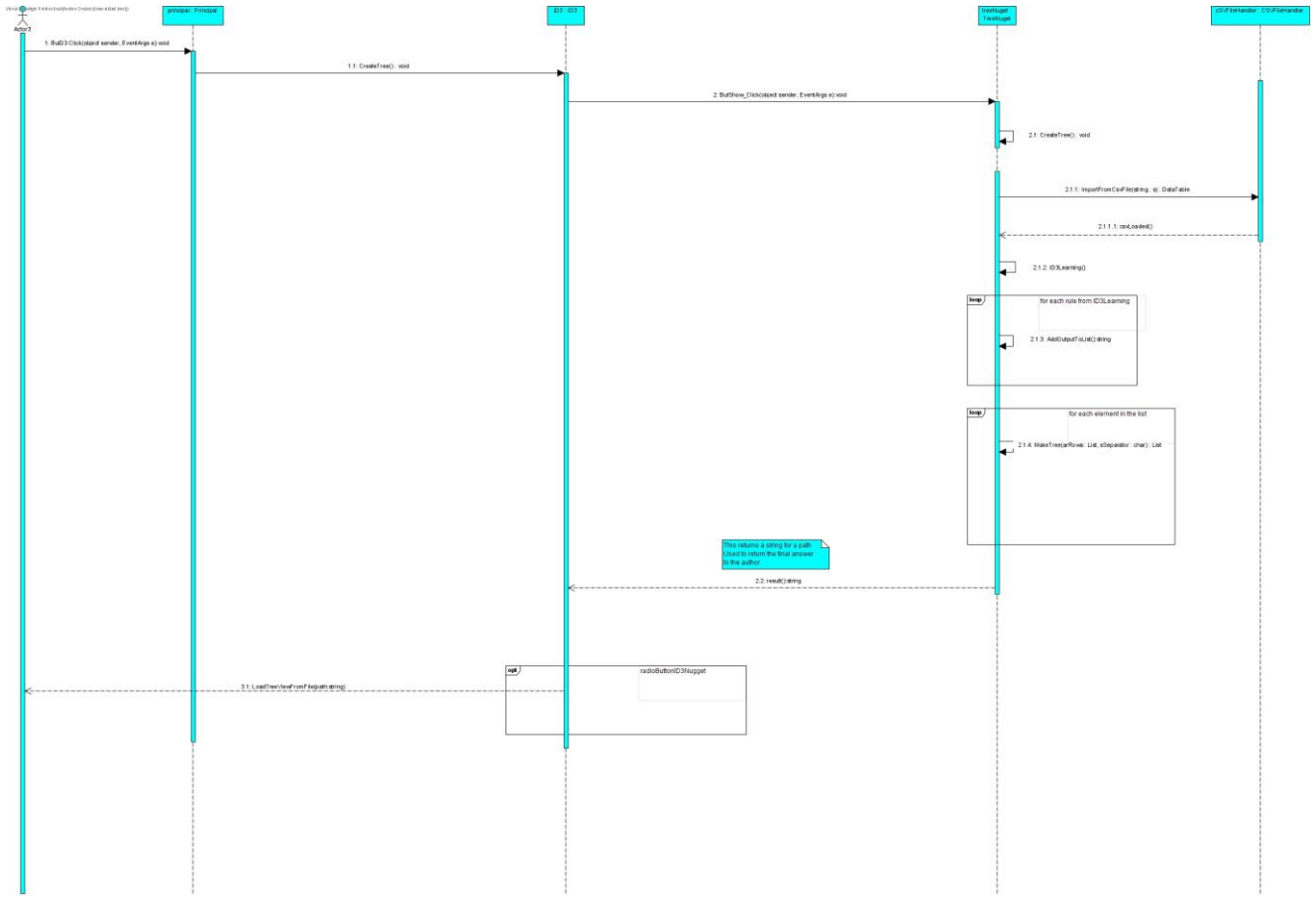

Decision_Tree
Tree
-Root : TreeNode
-Visual : string
+Print(string : TreeNode) : string
+saveStatus(string) : void
+CalculateResult(TreeNode : string, string : result, valuesForQuery : IDictionary) : string
+Learn(string : DataTable) : TreeNode
-CheckIfaTreeNode : Te(DataTable,string) : boolean
-CreateSmallerTable(DataTable : ds,string,s, i : int) : DataTable
-GetRootNode(ds : DataTable, s : string) : TreeNode
-GetGainForAllAttributes(ds : DataTable,int,s:double) : double
+CalculateTableEntropy()
+GetAmountOfEdgesAndTotalPositivResults(ds : DataTable,i:int) : List<int>
+CountKnownValues(ds : DataTable, i : int) : IEnumerable<String>
-ChildNodes
TreeNode
-Name : string
-Edges : string
-TableIndex : int
-isLeaf : bool
+TreeNode(in : string, ti : int, Na : Attribute, e : String)
+TreeNode(ti : bool,n:string,e:string)
-TreeNode
Attribute
-Name : string
-DifferentAttributeNames : List<String>
-InformationGain : double
+getName() : string
+setName(Name : string) : void
+getDifferentAttributeNames() : List<String>
+setDifferentAttributeNames(DifferentAttributeNames : List<String>) : void
+getInformationGain() : double
+setInformationGain(informationGain : double) : void
+GetDifferentAttributeNamesOfColumns(data : DataTable,columnIndex:int)
+Attribute(string : name,e DifferentAttributeNames : List)
-tree
Model
Car
-Brand : string
-Sold_Price : string
-Number_Doors : string
-Number_Occupants : string
-Lug_Boot_Size : string
-Safety_Level : string
+Car(string : c, sp : string, nd : string, lb : string, sl : string)
0..*
-cars
1
ListCars
-Cars : List<Cars>
-ListCars()
+Add_Car(string : c, sp : string, nd : string, lb : string, sl : string) : void
+Get_Range_From_Values() : int, ti : int, c : int) : List<Car>
+Get_SubString_For_A_Search(s : string) : List<Car>
0..*
-lc
GUI
ID3
+PATH : string
+TREE : string
+csv : CSVFileHandler
+tree : Tree
+path : string
+result : string
+ID3(path : string)
+CreateTree() : void
+ProbeTree() : void
+MakeTree(arRows : List<string>)
+LoadTreeViewFromTextfile_name= string) : void
-butProbe_Click(object, EventArgs) : void
-butRegister_Click(object, EventArgs) : void
Program
+Main() : void
<<use>>
Principal
+PATH : string
-lc : string
-ds :DataSet
-_categories : string[]
+Principal()
+Load_Data() : void
-InitializeStrings() : void
-Fill() : table(LastCar<Car>,lc):void
-butAll_Click(object sender, EventArgs e) : void
-butLoad_Click(object sender, EventArgs e) : void
-butAcepter_Click(object sender, EventArgs e) : void
+Search_Method() : void
+interval_Method(type : int) : void
-CategoriesCB_SelectedIndexChanged(object sender, EventArgs e) : void
-butStat_Click(object sender, EventArgs e) : void
-butTree_Click(object sender, EventArgs e) : void
-id3
-csv
CSVFileHandler
+CSVFileHandler()
+ImportFromCsvFile(string : s) :DataTable
+ExportToCsvFile(string : s) : void
-probe
-es
-c
Probe
-result : string
-ds : dataSet
-PATH : string
+Probe(result : string)
+CreateTable()
+CreateTree()
EnterSample
+tree : Tree
+buying : string[]
+maint : string[]
+doors : string[]
+persons : string[]
+lugboot : string[]
+safety : string[]
+EnterSample() : Tree)
+Fill(sp : string[], c : ComboBox) : void
+CalculateAOutput() : void
-butCal_Click(object sender, EventArgs e) : void
Graphics
-cars : List<Car>
-carIdadMax : int
+Graphics(cars : List<Car>)
+CreateChart() : void
-butPastelFor_Click(object sender, EventArgs e)
-numberOfDoors_Click(object sender, EventArgs e) : void
-numberOfPeople(object sender, EventArgs e) : void
-bumSize(object sender, EventArgs e) : void
-priceOfCars_Click(object sender, EventArgs e) : void
+carSoldPrice() : List<int>
+carsByMaintancePrice() : List<int>
+numberCarsPerNumberOfDoors() : List<int>
+numberOfCarsPerNumberOfPeople() : List<int>
+clear() : void
-noShow() : void
Decision_TreeNuget
TreeNuget
+NUGET : string
+PROBE : string
-path : string
-data : DataTable
-pb : DataTable
-csv : CVSFileHandler
-tre : DecisionTree
-result : string
-Codification : codebook
+TreeNuget(path : string)
+GetResult() : string
+CreateTree() : void
+Probe() : void
+Example(string : buying, maint : string, doors : string, persons : string, lug : string, safety : string)
+MakeTree(arRows : List, sSeparator : char) : List
<<use>>

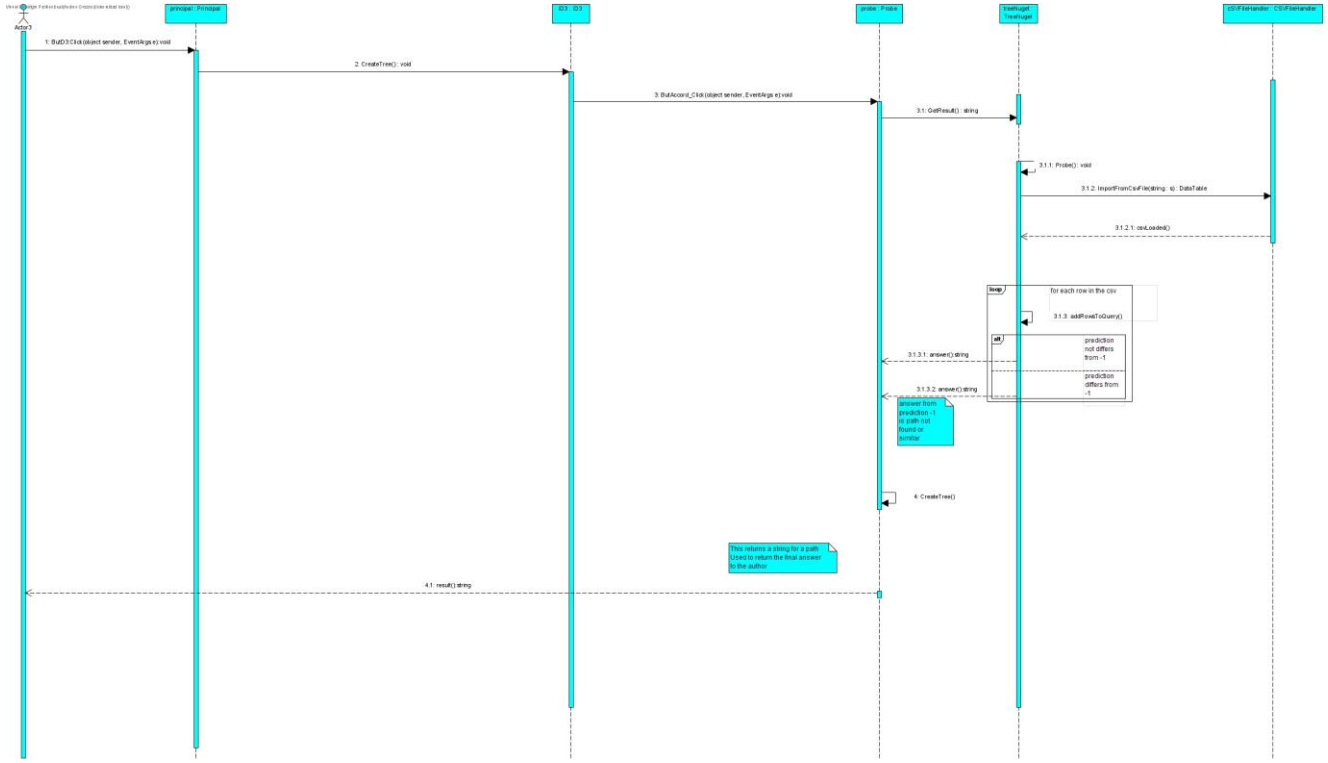

ListCars:
Cars
List
Car
Brand= "Renault"
SoldPrice= "vhigh"
MaintancePrice = "vhigh"
Number_Doors="4"
Number_Occupants="4"
Lug_Boot_Size="small"
Safety_Level="med"
Car
Brand= "Ford"
SoldPrice= "low"
MaintancePrice = "med"
Number_Doors="4"
Number_Occupants="4"
Lug_Boot_Size="small"
Safety_Level="med"
Car
Brand= "Chevrolet"
SoldPrice= "high"
MaintancePrice = "med"
Number_Doors="4"
Number_Occupants="4"
Lug_Boot_Size="big"
Safety_Level="med"
Car
Brand= "Peugeout"
SoldPrice= "vhigh"
MaintancePrice = "vhigh"
Number_Doors="2"
Number_Occupants="2"
Lug_Boot_Size="small"
Safety_Level="med"
Car
Brand= "Renault"
SoldPrice= "med"
MaintancePrice = "low"
Number_Doors="2"
Number_Occupants="4"
Lug_Boot_Size="small"
Safety_Level="low"


User
principal : Principal
listCars : ListCars
graphics : Graphics
1: butLoad_Click(object sender, EventArgs e):void
1.1: Load_Data() : void
2: CreateTable()
alt
buying(Category)
3: butAceptar_Click(object sender, EventArgs e) : void
3.0.1: Filter
maintance(Category)
3.1: butAceptar_Click(object sender, EventArgs e) : void
3.1.1: Filter
persons(Category)
3.2: butAceptar_Click(object sender, EventArgs e) : void
3.2.1: Get_Range_From_Values(i : int, d : int, c : int) : List<Car>
3.2.2: Interval_Method(type : int) : void
3.2.3: Filter
doors(Category)
3.3: butAceptar_Click(object sender, EventArgs e) : void
3.3.1: Get_Range_From_Values(i : int, d : int, c : int) : List<Car>
3.3.2: Interval_Method(type : int) : void
3.3.3: Filter
lugboot(Category)
3.4: butAceptar_Click(object sender, EventArgs e) : void
3.4.1: Filter
safety(Category)
3.5: butAceptar_Click(object sender, EventArgs e) : void
3.5.1: Filter
4: butGraph_Click(object sender, EventArgs e) : void
4.1: CreateChart() : void
alt
Buying(Diagram)
4.1.1: priceOffCars_Click(object sender, EventArgs e) : void
Maintance(Diagram)
4.1.2: butPastelPor_Click(object sender, EventArgs e)
Doors(Diagram)
4.1.3: numberOfDoors_Click(object sender, EventArgs e) : void
Occupants(Diagram)
4.1.4: numberOfPeople(object sender, EventArgs e) : void
Trunsize(Diagram)
4.1.5: trunkSize(object sender, EventArgs e) : void
4.2: showGraphic


User enter a Csv with the required data
Actor
Principal
ID3 : ID3
cSVFileHandler : CSVFileHandler
tree : Tree
1: buID3_Click(object sender, EventArgs e): void
1.1: CreateTree(): void
loop
1.1.1: ImportFromCsvFile(string : s) : DataTable
1.1.2: csvLoaded()
2: Learn(string : DataTable) : TreeNode
loop
alt
Is leaf
2.1: CheckIfis(TreeNode : Ts;DataTable:string): boolean
Is not leaf
2.2: Learn(string : DataTable) : TreeNode
2.3: Tree.root()
3: Print(string : TreeNode) : string
opt
Child Nodes are null
loop
3.1: saveStatus(string) : void
loop
For each Child Node in Node
3.2: Print(string : TreeNode): string
3.3: Print()
4: butShow_Click(object sender, EventArgs e): void
4.1: LoadTreeViewFromFile(string path): void
Returns a CSV with the modifieddata aput it in the tree view


Actor2
probe : Probe
ID3 : ID3
tree : Tree
1: ProbeTree()
1.1: ReadCSV()
loop
for each line in the CSV
1.2: CalculateResult(TreeNode : string, string : result, valuesForQuery : IDictionary) : string
alt
root is leaf
1.2.1: result()
Result is a chain of character AKA string
1.2.2: return result
root is not leaf
loop
For each Child Node in Root
loop
For each entry in values query
opt
Root Value equals to entry and child node edge equals to entry
1.2.2: removeEntry()
1.2.3: CalculateResult(TreeNode : string, string : result, valuesForQuery...
1.2.4: return result
Returns the path in a string that indicates the result of the decision tree
In other word returns the leafs of the decision tree
2: butProbe_Click(object Sender, EventArgs e) : void
2.1: CreateTable()
2.2: CreateTree()


principal : Principal
ID3 : ID3
treeNugget : TreeNugget
CSVFileHandler : CSVFileHandler
1: ButID3Click(object sender, EventArgs e) void
1.1: CreateTree() : void
2: ButShow_Click(object sender, EventArgs e) void
2.1: CreateTree() : void
2.1.1: ImportFromCsvFile(string : s) : DataTable
2.1.1.1: csvLoaded()
2.1.2: ID3Learning()
loop
for each rule from ID3Learning
2.1.3: AddOutputToList() string
loop
for each element in the list
2.1.4: MakeTree(arRows : List, sSeparator : char) : List
This returns a string for a path Used to return the final answer to the author
2.2: result() string
opt
radioButtonID3Nugget


Actor3
principal : Principal
f03 : f03
probe : Probe
treeNuget : TreeNuget
clsFileHandler : ClsFileHandler
1: ButD3:Click(object sender, EventArgs e):void
2: CreateTree(): void
3: ButAccord_Click(object sender, EventArgs e):void
3.1: GetResult() : string
3.1.1: Probe(): void
3.1.2: ImportFromCsvFile(string: s) : DataTable
3.1.2.1: csvLoaded()
loop
for each row in the csv
3.1.3: addRowsToQuery()
alt
prediction not differs from -1
prediction differs from -1
3.1.3.1: answer():string
3.1.3.2: answer():string
answer from prediction -1 is path not found or similar
4: CreateTree()
This returns a string for a path. Used to return the final answer to the author
4.1: result():string

Referencias

Aaron, T. (2010). *Universidad Tecnologica de la Mixteca.* Obtenido de UTM: http://www.utm.mx/~jahdezp/archivos%20estructuras/DESICION.pdf

Na8. (13 de Abril de 2018). *Aprende Machine Learning.* Obtenido de https://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/

Universidad de Barcelona. (2008). *Universitat de Barcelona*. Obtenido de Universitat de Barcelona: http://www.ub.edu/aplica_infor/spss/cap2-3.htm