Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

LICENCIATURA EN COMPUTACION

# PROYECTO DE INVESTIGACIÓN II

## Minería de Datos para el análisis de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico


Asesor de Proyecto de Investigación

Dr. Benjamín Moreno Montiel

_____

Alumnos

Flores Morales Yafte Aaron 2123041370

Mandujano López Karla 2123010080


Diciembre del 2016

# Agradecimientos

A nuestro asesor Dr. Benjamín Moreno Montiel por compartirnos sus conocimientos, dedicarnos parte de su tiempo y apoyándonos en todo momento.

A la Dra. Miriam Moreno Montiel por habernos recibido en su casa de estudios y brindarnos parte de su tiempo.

# Contenido

# Lista de Figuras

# Lista de Tablas

# Introducción

## 1.1. Introducción

El petróleo crudo puede tener grandes usos y aplicaciones, para lograr esto se debe someter a un proceso de conversión de energía primaria a secundaria denominado refinación. La refinación es el conjunto de procesos que se aplican al petróleo crudo con la finalidad de separar sus componentes útiles y adecuar sus características a las necesidades de la sociedad.

El problema es el siguiente, se alimenta al reactor FCC (Fluid Cracking Catalytic) con una corte de petróleo que se denomina gasóleo y mediante una serie de reacciones complejas y con ayuda de un catalizador (sustancia que favorece una reacción química) se obtienen productos de mayor valor económico que el gasóleo alimentado. Entre estos productos se encuentra la gasolina el cual es el producto que nos interesa estudiar en este Proyecto Terminal.

Por lo cual se busca implementar un programa de evaluación que garantice la obtención de resultados que reflejen el desempeño real de las simulaciones de los catalizadores evaluados y así proceder a la elección de aquel que cumpla con los requerimientos industriales de este proceso o proponer mejoras a este en función de su comportamiento en el proceso. Sin embargo, debido al gran número de componentes presentes, caracterizar y describir a nivel molecular la complejidad de la composición de las cargas de alimentación empleadas en las plantas de FCC y el de sus productos representa un esfuerzo poco viable con las técnicas empleadas para el modelamiento cinético.

Para evitar estos problemas, es por ello que se quiere llevar a cabo el proceso de Minería de Datos sobre el repositorio de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico, obteniendo una simulación que nos de buenos resultados.

## 1.2. Hipótesis planteada

Se pretende llevar a cabo el proceso de Minería de Datos sobre un repositorio de datos de un Reactor de Microactividad de Cracking Catalítico para obtener patrones, conjunto de atributos que definen un comportamiento especial, que nos aporten beneficios al proceso de simulación.

# 1.3. Objetivos

## 1.3.1. Objetivo General

Llevar a cabo el proceso de Minería de Datos para el análisis de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico.

## 1.3.2. Objetivos Particulares

- Revisar los principales aspectos teóricos necesarios para realizar el análisis de datos de un Reactor de Microactividad de Cracking Catalítico.

- Aplicar las fases de extracción de conocimiento sobre el repositorio del Reactor de Microactividad de Cracking Catalítico, ya que los datos se encuentran en un estado crudo.
- Hacer la revisión de los principales clasificadores del aprendizaje maquinal que mejor se ajuste sobre el repositorio del Reactor de Microactividad de Cracking Catalítico.

- Implementación de clasificadores que nos permitirá llevar a cabo el proceso de clasificación sobre los datos, para realizar el análisis de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico.

- Calculo de las medidas de funcionamiento para evaluar que tan bueno son los clasificadores implementados.

- Análisis el comportamiento de los resultados finales para obtener las conclusiones finales de Proyecto Terminal.

# 1.4. Justificación

La minería de datos ha sido una de las áreas que ha tenido grandes avances en los últimos años, debido al incremento en el tamaño de las bases de datos. Por lo regular siempre se hace el análisis con repositorios que se encuentran en el área de investigación de Ciencias y tecnologías de Información, pero hoy en día los proyectos de investigación interdisciplinarios son los más rentables.

Se debe realizar una búsqueda en otras áreas del conocimiento para realizar un proceso de exploración de datos mucho más relevante o que tenga un mayor imparto en el proceso en el que se tratan los datos de estas otras áreas de investigación.

Es por ello que una de las principales justificaciones de la realización de este proyecto radica en que se han realizado muy pocos estudios en los que combinen estas dos áreas de conocimiento, las Ciencias y Tecnologías de la Información y la Ingeniería Química.

El modelamiento de FCC por lo regular se basa en el empleo de agrupamiento de especies y la construcción de modelos matemáticos que describan al reactor desde simples a complejos, lo cual incide directamente en los tiempos empleados para la predicción. Sin embargo, muchos de estos modelos solamente son confiables bajo las condiciones de operación empleadas.

Por ello se pretende llevar a cabo el proceso de Minería de Datos sobre un repositorio de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico para obtener patrones que nos aporten beneficios al proceso de simulación.

# Antecedentes

## 2.1. Aprendizaje Maquinal

Una de las cosas que busca la Inteligencia Artificial es desarrollar programas para resolver problemas que tienen un grado de dificultad, pero en varias ocasiones no se puede obtener la solución óptima ya que no se pueden explorar todas las alternativas y en dado caso solo se obtienen soluciones parciales con buenos resultados.

Otra cosa que se busca es crear programas que conforme sean utilizados sean capaces de adaptarse e integrar nuevo conocimiento, con lo cual al presentarle un nuevo problema a este sea capaz de resolverlo sin necesidad de modificarlo.

El Aprendizaje Maquinal es un área de la Inteligencia Artificial la cual tiene como objetivo el desarrollo de programas que puedan adaptar su comportamiento de manera autónoma basados en su experiencia.

El Aprendizaje Maquinal se divide en cuatro tipos de aprendizaje:

- **Aprendizaje inductivo:** Se crean modelos de conceptos que posean características comunes de los ejemplos de entrenamiento.

  En el aprendizaje inductivo existen dos tipos:
  - *Aprendizaje supervisado:* En este aprendizaje viéndolo desde el enfoque de clasificación se tiene un conjunto de ejemplos con un atributo asociado llamado clase.

  En la tabla 2.1 se muestra el aprendizaje supervisado.

| Atributo1 | Atributo2 | … | AtributoN | Clase |
|---|---|---|---|---|
| 1594 | 5874 | | 4651 | X |
| 3645 | 2254 | | 2643 | Y |
| 3245 | 6651 | | 3355 | X |
| 0697 | 2665 | | 9855 | X |
| 3647 | 2654 | | 3014 | Y |

Tabla 2.1 Estructura de una Base de Datos del Aprendizaje Supervisado

  Más adelante se revisará en este trabajo métodos de clasificación para este tipo de aprendizaje como son Bayes Ingenuo, K vecinos más cercanos e ID3.

  - *Aprendizaje no supervisado:* En este tipo de aprendizaje también se tiene un conjunto de ejemplos, pero se desconoce la clase de cada atributo.

  En la tabla 2.2 se muestra el aprendizaje no supervisado.

| Atributo1 | Atributo2 | ... | AtributoN |
|---|---|---|---|
| 1594 | 5874 | | 4651 |
| 3645 | 2254 | | 2643 |
| 3245 | 6651 | | 3355 |
| 0697 | 2665 | | 9855 |
| 3647 | 2654 | | 3014 |

Tabla 2.2 Estructura de una Base de Datos del Aprendizaje no Supervisado

- **Aprendizaje analítico o deductivo:** Se aplica la deducción para obtener descripciones generales a partir de un ejemplo de concepto y su explicación.
- **Aprendizaje genético:** Se aplican algoritmos inspirados en la teoría de la evolución para encontrar descripciones generales a conjuntos de ejemplos.
- **Aprendizaje conexionista:** Se buscan descripciones generales mediante el uso de la capacidad de adaptación de redes de neuronas artificiales.

Algunos de los usos que tiene el Aprendizaje Maquinal son los siguientes tres:

- **Tareas difíciles de programar:** Existen tareas muy complejas en las cuales hacer un programa para resolverlas es muy difícil como por ejemplo adquisición de conocimiento, reconocimiento de rostros, de voz, entre otras. Estas últimas tareas se pueden resolver con el Aprendizaje Maquinal realizando una clasificación a partir de un conjunto de ejemplos.

- **Aplicaciones autoadaptables:** Estas aplicaciones son capaces de ir aprendiendo conforme van obteniendo el conocimiento e irse adaptando a las circunstancias. Algunos ejemplos son las interfaces inteligentes, filtros de spam, sistemas recomendadores, etc.

- **Minería de datos:** El Aprendizaje Maquinal nos sirve en la extracción de conocimiento sobre grandes cantidades de datos, obteniendo patrones no triviales que nos permitan obtener conocimiento novedoso y potencialmente útil. En minería de datos se utilizan algoritmos de clasificación del aprendizaje maquinal conocidos como clasificadores para realizar dicha extracción.

Esta tarea del Aprendizaje Maquinal será explicada con mayor detalle en la siguiente sección, ya que es nuestro interés en este Proyecto Terminal

## 2.2. Minería de Datos

La minería de datos ha tenido un gran avance los últimos años, esto debido a que las bases de datos han incrementado su tamaño, con lo cual se tiene mucho conocimiento implícito que es necesario explorar y analizar para intentar descubrir patrones en tan grandes volúmenes de conjuntos de datos.

La minería de datos es la extracción de conocimiento sobre grandes cantidades de datos. El conocimiento extraído debe contar con las características de no ser trivial, implícito, previamente desconocido y potencialmente útil, esto para poder obtener patrones no triviales que nos permitan obtener conocimiento novedoso y potencialmente útil.

El patrón es un conjunto de atributos que definen un comportamiento especial en la base de datos que se está utilizando.

Para saber cuándo se están manejando grandes cantidades de datos contamos con la Taxonomía de Huber-Wegman que se muestra en la Tabla 2.3 en la cual nos muestra de que tamaño es la base de datos.

| Clasificación | Tamaño de la Base d Datos |
|---|---|
| Tiny | <=100 registros |
| Small | <=1000 registros |
| Medium | <=10,000 registros |
| Large | <=500,000 registros |
| Huge | <=10,000,000 |
| Massive | 10,000,000 |

Tabla 2.3 Taxónomia de Huber- Wegman

La forma más común en que visualizamos una base de datos en Minería de Datos es en forma de una tabla de N*M, donde N es el número de ejemplos o registros y M son el número de atributos que componen la base de datos, como se muestra en la Tabla 2.4.

M

| Dia | Pronóstico | Temperatura | Humedad | Viento | Clase |
|---|---|---|---|---|---|
| D1 | Soleado | Calor | Alta | Débil | No |
| D2 | Soleado | Calor | Alta | Fuerte | No |
| D3 | Nublado | Calor | Alta | Débil | Si |
| D4 | Lluvia | Templado | Alta | Débil | Si |
| D5 | Lluvia | Frío | Normal | Débil | si |
| D6 | Lluvia | Frío | Normal | Fuerte | no |

N

Tabla 2.4 Visualización de una Base de Datos en Minería de Datos

Cada ejemplo de la Tabla 2.4 está compuesto por M atributos que pueden ser de dos tipos:

- Discreto: Un atributo discreto tiene un número finito de valores.
- Continuo: Un atributo continuo tiene un número infinito de valores.

Existen 5 tareas que nos permiten llevar a cabo la minería de datos las cuales son:

- **Predicción:** La predicción nos permites descubrir la manera de como ciertos atributos dentro de los datos se comportan en el futuro.
- **Identificación:** Identificar la existencia de objetos, eventos y actividades dentro de los datos.
- **Agrupamiento:** Nos permite maximizar las similaridades y minimizar las diferencias entre los objetos, mediante algún criterio de agrupamiento.
- **Asociación:** Las reglas de asociación intentan descubrir cuáles son las conexiones que se pueden tener entre los objetos identificados.
- **Clasificación:** Mediante esta tarea podemos separar los datos de acuerdo a las clases que sean asignadas a cada ejemplo en los datos.

Esta última tarea llamada Clasificación es la que se llevara a cabo en la Minería de Datos para el análisis de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico Lo que se pretende hacer con la clasificación es la separación de un conjunto de datos acorde a una clase predicha por el modelo de clasificación. Esto se hará mediante distintos clasificadores los cuales son Bayes Ingenuo, K vecinos más cercanos e ID3.

## 2.3. Extracción de conocimientos en base de datos

La extracción de conocimiento está principalmente relacionada con el proceso de descubrimiento ya que, con miles de datos necesitamos limpiarlos (eliminar fragmentos innecesarios, repetidos, etc.), organizarlos y una vez realizado este proceso decimos que tenemos información. La información hay que tratarla con un modelo para obtener resultados o conclusiones a los que llamamos conocimiento. Es decir, el conocimiento es información analizada.

El objetivo principal de la extracción de conocimiento de base de datos (KDD - Knowledge Discovery in Databases) es el desarrollo de métodos y técnicas para encontrar dicho conocimiento en las bases de datos.

Una vez que se obtiene el conocimiento de los datos, estos tienen un valor real ya que puede ayudarnos a tomar mejores decisiones a una serie de problemas.

El KDD tiene una serie de metas, las cuales son:

- Se deben procesar grandes cantidades de datos de manera automática.
- Identificar dentro de estas grandes cantidades de datos cuales van a ser los patrones más significativos y relevantes.
- La interpretación de los patrones para poder representarlos como conocimiento.

Las fases del KDD se muestran en la figura 2.1:

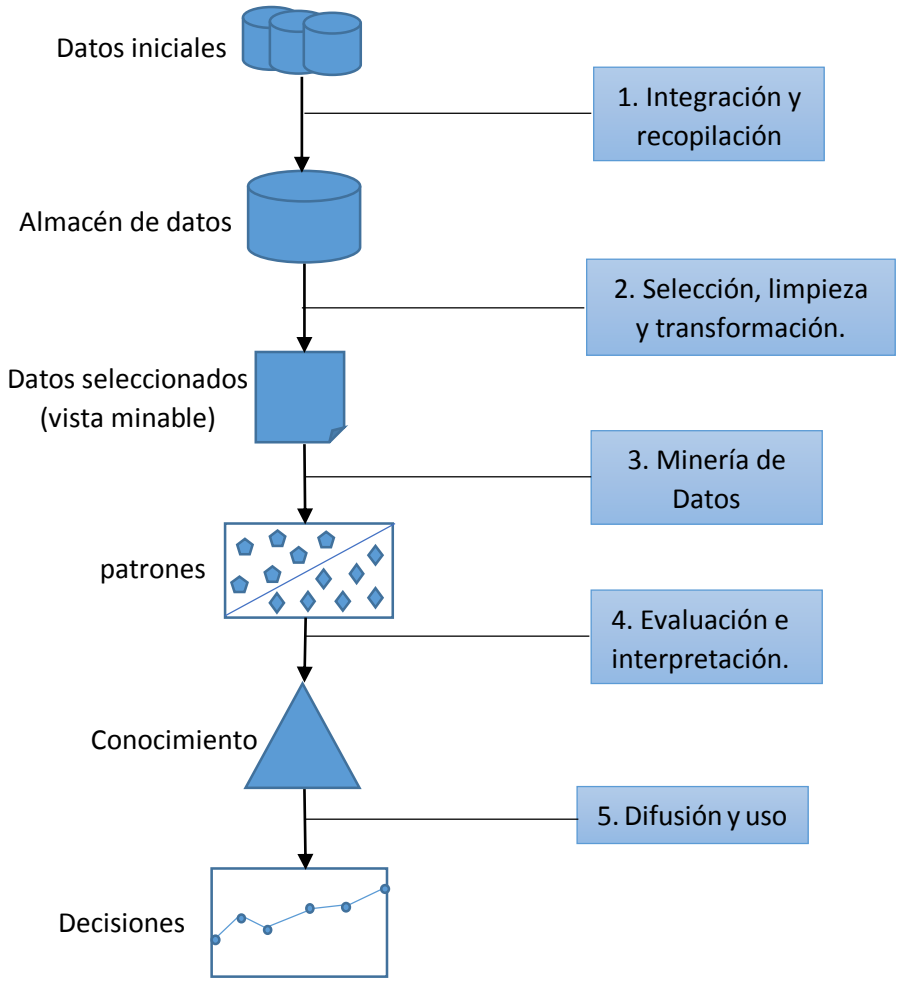



Figura 2.1 Fases del KDD

Estas fases son los pasos para obtener el conocimiento de las bases de datos. Las primeras fases del KDD determinan que las fases sucesivas sean capaces de extraer conocimiento válido y útil a partir de la información original.

Las fases consisten en lo siguiente:

1. **Integración y recopilación**: Esta fase es la primera que se hace, ya que por lo general en la vida real no se tienen los datos organizados en una tabla o archivo, sino que estos se encuentran dispersos en diferentes documentos o incluso hasta en hojas de papel y se deben recopilar e integrar todos estos para poder pasar a la siguiente fase.

2. **Selección, limpieza y transformación**: Primero se realiza la limpieza ya que se busca encontrar los atributos que nos aporten el mayor beneficio, así como eliminar aquellos que no nos aportan beneficios al problema en cuestión. Una vez que tenemos una base de datos limpia podemos pasar a la selección, esta fase sirve para encontrar los datos relevantes para el problema, es decir, se identifica la clase que distingue a cada uno

de los ejemplos. Esto es de suma importancia ya que es necesaria para la minería de datos. En dado caso que no se pueda, entonces se aplican técnicas de agrupamiento. Por último, se aplica la transformación en la cual se debe definir el formato de los datos para poder aplicar los métodos de la Minería de Datos.

3. **Minería de Datos:** En esta fase se aplican los métodos de extracción de patrones y son calculadas las medidas de funcionamiento, las cuales sirven para evaluar el que tan bueno es el modelo de clasificación.

4. **Evaluación e interpretación:** Es aquí donde se analizan los patrones obtenidos en la fase anterior, para poder resolver la problemática inicial. Por lo cual se busca encontrar el conocimiento.

Vamos a presentar un ejemplo en el cual aplicamos el proceso KDD sobre datos de un Reactor de Microactividad de Cracking Catalítico.

## 2.4. Proceso KDD sobre datos de un Reactor de Microactividad de Cracking Catalítico

En esta subsección vamos a analizar el proceso KDD que se realizó para obtener la base de datos final con la cual se va a trabajar para la obtención de conocimiento.

Fases del porceso KDD:

1. **Integración y recopilación**: No contamos con todos los datos organizados en una tabla o un solo archivo, sino que estos se encentran dispersos en diferentes documentos, por lo cual los recopilamos.

En un inicio se tenían 3 archivos Excel llamados:

- estima_para_IMP
- Data_PROTOC_ACE_Masa_cataliz
- exp_imp_doctorado

Y un archivo Word llamado:

- Anexos

La vista de los archivos Excel se muestra en la figura 2.2:

| | A | B | C |
|---|---|---|---|
| 5 | | | |
| 6 | Corrida No. | 107 | 108 |
| 7 | Fecha de evaluacion | 19/03/2001 | 19/03/2001 |
| 8 | Identificador del catalizador | **Ecat+FILLER** | **Ecat+FILLER** |
| 9 | Nombre del catalizador | **IMPFCC-51** | **IMPFCC-51** |
| 10 | Identificacion de la carga | CARGA TIPICA | CARGA TIPICA |
| 11 | Nombre de la carga | TULA-1 | TULA-1 |
| 12 | peso carga alimetnada, g | 0.6000 | 0.6000 |
| 13 | Temperatura reaccion, °C | 520.0 | 520.0 |
| 14 | Tiempo inyeccion, s | **30.0** | **30.0** |
| 15 | Tiempo agot. catalizador, s | 230 | 230 |
| 16 | tiempo agot. liquido, s | 240 | 240 |
| 17 | Posicion inyector, pulg | 1.125 | 1.125 |
| 18 | CCR incorporado por carga | 0.660 | 0.660 |
| 19 | Recuperacion, % peso | 96.0 | 97.4 |
| 20 | Relacion C/O, g/g | 3.0 | 4.0 |
| 21 | Conversion, %peso | 58.09 | 62.68 |
| 22 | Peso de catalizador , g | 1.8 | 2.4 |
| 23 | Flujo de carga, g/min | 1.2 | 1.2 |
| 24 | Peso de producto liquido, g | 0.4121 | 0.3924 |
| 25 | Peso total de gas (con N2) | 0.7758 | 0.7953 |
| 26 | mg coque | 27.5829 | 32.5027 |
| 27 | mg coque / g catalizador | **15.3238** | **13.5428** |
| 28 | | **1.8276** | **2.4325** |
| 29 | K | **1.3859** | **1.6796** |
| 30 | Rendimientos, %Peso: | | |
| 31 | Coque | 4.789 | 5.561 |
| 32 | Gas seco | 1.323 | 1.527 |
| 33 | Hidrogeno | 0.094 | 0.097 |
| 34 | H2S | 0.000 | 0.000 |
| 35 | Metano | 0.381 | 0.444 |
| 36 | Etano | 0.255 | 0.294 |
| 37 | Etileno | 0.593 | 0.692 |
| 38 | Propano | 0.661 | 0.882 |
| 39 | Propileno | 3.933 | 4.407 |
| 40 | n-Butano | 0.551 | 0.720 |
| 41 | Isobutano | 2.749 | 3.495 |
| 42 | C4 Olefinas | 4.394 | 4.560 |
| 43 | 1-Buteno | 0.951 | 0.994 |
| 44 | Isobutileno | 1.262 | 1.264 |
| 45 | c-2-Buteno | 0.911 | 0.962 |
| 46 | t-2-Buteno | 1.253 | 1.323 |
| 47 | Butadieno | 0.018 | 0.017 |
| 48 | Gasolina | 39.531 | 41.361 |
| 49 | LCO | 19.098 | 18.589 |



Figura 2.2 Forma cruda de los datos contenidos en archivos Excel

En la Figura 2.2 se puede observar que estos datos tienen el formato utilizado en Ingeniería Química, los atributos están colocados de forma vertical y que están organizados para el formato del área de conocimiento, por lo cual se tienen los datos dispersos y algunos ellos no nos sirven para los fines de este proyecto.

De igual forma el archivo Word contiene tablas similares las cuales se ven como la tabla 2.5.

| NUMERO DE CORRIDA | 1 | 2 |
|---|---|---|
| Temperatura de reacción, °C | 520 | 520 |
| Tiempo inyección, s | 120 | 45 |
| Tiempo agot. catalizador, s | 350 | 305 |
| Tiempo agot. líquido, s | 840 | 315 |
| Posición inyector, pulg | 1.125 | 1.125 |
| Recuperación, % peso | 98.08 | 98.27 |
| Relación C/O, g/g | 3.75 | 10 |
| Conversión, %peso | 65.24 | 72.84 |
| Peso de catalizador , g | 9 | 9 |

| | | |
|---|---|---|
| **Flujo de carga, g/min** | 1.2 | 1.2 |
| **Peso de producto líquido, g** | 1.5904 | 0.5318 |
| **Peso total de gas (con $N_2$)** | 2.8172 | 1.1952 |
| **RENDIMIENTOS, %PESO:** | | |
| **Coque** | 3.3365 | 6.5070 |
| **Gas seco** | 1.7696 | 2.2303 |
| **Hidrogeno** | 0.1544 | 0.2268 |
| **H2S** | 0.0000 | 0.0000 |
| **Metano** | 0.5604 | 0.7469 |
| **Etano** | 0.4485 | 0.5002 |
| **Etileno** | 0.6062 | 0.7565 |
| **Propano** | 0.9851 | 1.1709 |
| **Propileno** | 4.2668 | 4.8810 |
| **n-Butano** | 0.9602 | 1.1313 |
| **Isobutano** | 3.6480 | 4.4513 |
| **C4 Olefinas** | 5.2799 | 5.5214 |

Tabla 2.5 Forma cruda de los datos contenidos en el archivo Word

Estas tablas contienen los mismos atributos que los archivos Excel que en total son 48, por lo cual se va a tomar una porción más pequeña de esta tabla para explicar la siguiente fase del proceso KDD.

2. **Selección, limpieza y transformación**

   **Limpieza:** Como ya se mencionó se cuentan con 48 atributos, los cuales varios de ellos no son útiles para este trabajo de investigación, así que se eliminaran aquellos que no aportan nada.

   En la tabla 2.6 se observan los distintos datos que se tienen, antes de aplicar la limpieza.

| **NUMERO DE CORRIDA** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
|---|---|---|---|---|---|---|---|---|
| **Temperatura de reacción, °C** | 520 | 520 | 535 | 535 | 550 | 550 | 565 | 565 |
| **Tiempo inyección, s** | 120 | 45 | **120** | **45** | 120 | 45 | 120 | 45 |
| **Tiempo agot. catalizador, s** | 350 | 305 | 350 | 305 | 350 | 305 | 350 | 305 |
| **Tiempo agot. liquido, s** | 840 | 315 | 840 | 315 | 840 | 315 | 840 | 315 |
| **Posición inyector, pulg** | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 |
| **Recuperación, % peso** | 99.7 | 98.9 | 99.4 | 98.3 | 101.7 | 97.8 | 99.0 | 98.3 |
| **Relación C/O, g/g** | 3.8 | 10.0 | 3.8 | 10.0 | 3.8 | 10.0 | 3.8 | 10.0 |
| **Conversión, %peso** | 62.96 | 70.70 | 63.63 | 71.46 | 64.94 | 72.45 | 66.11 | 73.04 |
| **Peso de catalizador , g** | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 |
| **Flujo de carga, g/min** | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 |
| **Peso de producto liquido, g** | 1.6801 | 0.5517 | 1.6369 | 0.5279 | 1.6102 | 0.5037 | 1.5383 | 0.5023 |
| **Peso total de gas (con $N_2$)** | 2.7000 | 1.1113 | 2.7169 | 1.1261 | 2.7151 | 1.1273 | 2.7799 | 1.1485 |
| **RENDIMIENTOS, %PESO:** | | | | | | | | |
| **Coque** | 3.633 | 6.847 | 3.560 | 7.026 | 3.672 | 7.397 | 3.812 | 7.405 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Gas seco** | 1.723 | 2.140 | 2.094 | 2.560 | 2.418 | 2.943 | 3.145 | 3.553 |
| **Hidrogeno** | 0.153 | 0.216 | 0.163 | 0.241 | 0.166 | 0.239 | 0.218 | 0.281 |
| **H2S** | 0.000 | 0.0000 | 0.000 | 0.0000 | 0.000 | 0.0000 | 0.000 | 0.0000 |
| **Metano** | 0.550 | 0.711 | 0.723 | 0.914 | 0.890 | 1.117 | 1.200 | 1.408 |
| **Etano** | 0.447 | 0.483 | 0.533 | 0.566 | 0.613 | 0.645 | 0.789 | 0.762 |
| **Etileno** | 0.573 | 0.728 | 0.675 | 0.839 | 0.749 | 0.941 | 0.938 | 1.102 |
| **Propano** | 0.952 | 1.152 | 1.077 | 1.292 | 1.208 | 1.407 | 1.331 | 1.497 |
| **Propileno** | 3.948 | 4.707 | 4.373 | 5.175 | 4.822 | 5.593 | 5.239 | 5.966 |

Tabla 2.6 Base de Datos sin aplicación del proceso KDD

Se van a eliminar todos los atributos que se encuentran debajo del atributo RENDIMIENTOS %PESO, ya que atributos como: coque, Gas seco, Hidrógeno, etc. es el resultado que se obtiene después del proceso de refinación, por lo cual estos no nos sirven. Quedando una tabla más reducida como se muestra en la tabla 2.7

| **NUMERO DE CORRIDA** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
|---|---|---|---|---|---|---|---|---|
| **Temperatura de reacción, °C** | 520 | 520 | 535 | 535 | 550 | 550 | 565 | 565 |
| **Tiempo inyección, s** | 120 | 45 | **120** | **45** | 120 | 45 | 120 | 45 |
| **Tiempo agot. catalizador, s** | 350 | 305 | 350 | 305 | 350 | 305 | 350 | 305 |
| **Tiempo agot. liquido, s** | 840 | 315 | 840 | 315 | 840 | 315 | 840 | 315 |
| **Posición inyector, pulg** | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 | 1.125 |
| **Recuperación, % peso** | 99.7 | 98.9 | 99.4 | 98.3 | 101.7 | 97.8 | 99.0 | 98.3 |
| **Relación C/O, g/g** | 3.8 | 10.0 | 3.8 | 10.0 | 3.8 | 10.0 | 3.8 | 10.0 |
| **Conversión, %peso** | 62.96 | 70.70 | 63.63 | 71.46 | 64.94 | 72.45 | 66.11 | 73.04 |
| **Peso de catalizador , g** | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 | 9.00 |
| **Flujo de carga, g/min** | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 |
| **Peso de producto liquido, g** | 1.6801 | 0.5517 | 1.6369 | 0.5279 | 1.6102 | 0.5037 | 1.5383 | 0.5023 |
| **Peso total de gas (con $N_2$)** | 2.7000 | 1.1113 | 2.7169 | 1.1261 | 2.7151 | 1.1273 | 2.7799 | 1.1485 |

Tabla 2.7 Base de Datos pasada por el proceso de refinación

Atributos como Posición inyector, Peso de catalizador y Flujo de carga los eliminamos, ya que sus valores siempre son los mismo y no aporta una diferencia.

Los atributos Recuperación, Peso de producto líquido, Peso total de gas, son atributos que se obtienen después de realizar la refinación, por lo cual no son necesarios y los eliminamos, quedándonos solo con 6 atributos: Temperatura de reacción, Tiempo inyección, Tiempo agot. Catalizador, Tiempo agot. Líquido, Relación C/O y Conversión.

Una vez que se tiene la base de datos limpia podemos pasar a la siguiente fase de selección.

**Selección:** Esta fase nos sirve para encontrar los datos relevantes para los problemas, es decir, se identifica la clase que distingue a cada uno de los ejemplos, esta parte es esencial para poder realizar la minería de datos.

El atributo que seleccionamos como la clase es Conversión ya que este nos indica el peso del material obtenido, el cual dependiendo de este peso se podrá determinar si es bueno, medio o malo.

Para determinar esto se decidió por la siguiente escala:

Si
- Conversión <60% se considera **mala**
- 60<Conversión<70 se considera **media**
- 70<Conversión<80 se considera **buena**

**Transformación:** En esta tarea se define el formato de los datos para poder aplicar los métodos de la Minería de Datos.

Para poder manejar de una forma más sencilla estos atributos, los renombramos con unos nombres más cortos que los representaran de la siguiente forma:

- **Temperatura de reacción** → TempReaccion
- **Tiempo inyección** → TiempInyec
- **Tiempo agot. Catalizador** → TiempCata
- **Tiempo agot. Líquido** → TiempLiq
- **Relación C/O** → RelacionCO
- **Conversión** → Clase

Colocamos los atributos de forma horizontal, ya que los necesitamos de esta forma obteniendo una tabla con la estructura que se muestra en la Tabla 2.8

| TempReaccion | TiempInyec | TiempCata | TiempLiq | RelacionCO | Clase |
|---|---|---|---|---|---|
| 520 | 120 | 350 | 840 | 3.8 | 62.96 |
| … | … | … | … | … | … |

Tabla 2.8 Base de datos en la fase de Selección

Y como los valores de resultado deben ser un String y no un valor numérico, se cambian todos los valores numéricos por su correspondiente String. Obteniendo la Tabla 2.9

| TempReaccion | TiempInyec | TiempCata | TiempLiq | RelacionCO | Clase |
|---|---|---|---|---|---|
| 520 | 120 | 350 | 840 | 3.8 | media |
| … | … | … | … | … | … |

Tabla 2.9 Modificación de la clase

**Minería de Datos:** En esta fase se aplican los métodos de extracción de patrones y son calculadas las medidas de funcionamiento para evaluar que tan bueno es el modelo de clasificación. Los métodos que se utilizarán son Bayes Ingenuo, K vecinos más cercanos e ID3, que serán revisados más adelante.

# 2.5. Introducción de Ingeniería Química

A lo largo de la historia hemos ocupado los recursos de la naturaleza para satisfacer las nuestras necesidades de consumo, cada vez son mayores las necesidades para nosotros.
La ingeniería química busca satisfacer las necesidades diarias por medio del diseño, manutención, optimización, simulación y construcción de todo tipo de elementos en la industria de procesos, que es aquella relacionada con la producción de compuestos y productos cuya elaboración requiere de sofisticadas transformaciones físicas y químicas de la materia.
Algunas de las ramas de la ingeniería química son las siguientes:

- Ingeniería Química Industrial
- Ingeniería Química Petrolera
- Ingeniería en Metalurgia y Materiales

La ingeniería química petrolera combina métodos científicos y prácticos dirigidos al desarrollo de técnicas para descubrir, explotar, desarrollar y tratar los hidrocarburos desde su estado natural, en el yacimiento, hasta los productos finales o derivados
La ingeniería química petrolera, tienen muchas líneas de investigación, una de ellas es "CATALISIS", la catálisis comprende la síntesis, caracterización y evaluación de catalizadores para procesos de hidrotratamiento, desintegración catalítica y degradación de contaminantes en agua que responden a la solución de necesidades del país en la industria petrolera y ambiental.

La *desintegración catalítica* es un proceso muy importante en los sistemas de refinación, podemos definir la desintegración catalítica como un proceso de la refinación del petróleo que consiste en la descomposición de los componentes del petróleo en presencia de un catalizador para generar diferentes tipos de productos.

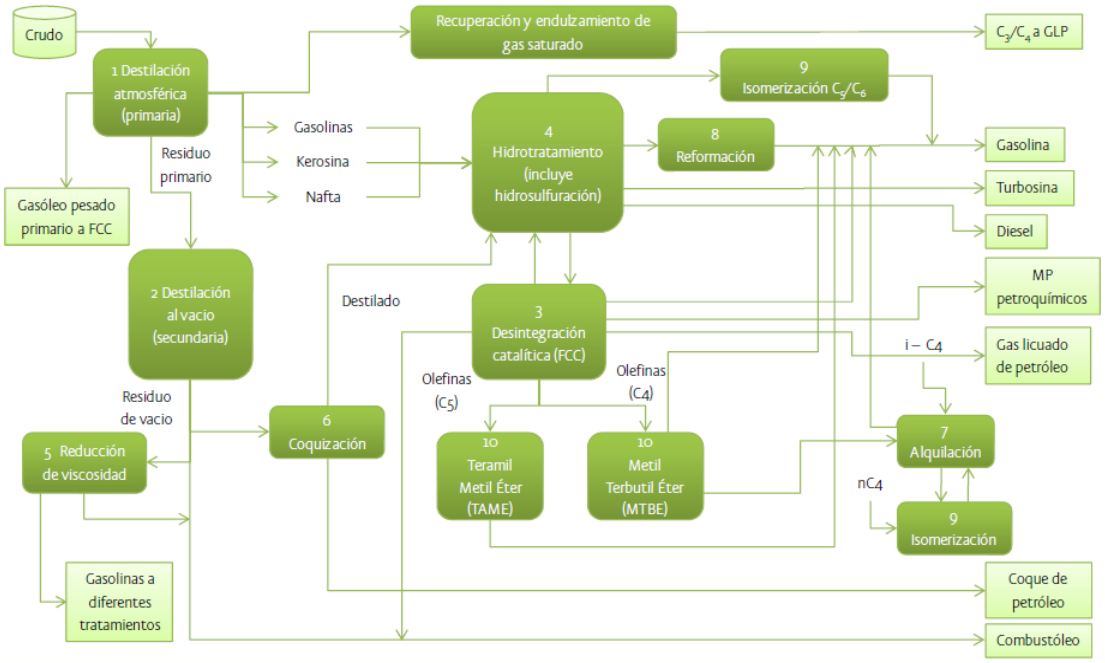



Figura 2.3 sistema de refinación

La figura 2.3 muestra un esquema general del sistema de refinación, la entrada principal (llamada también Carga) es el CRUDO, el cual sigue un proceso que descompone y transforma el crudo para producir diferentes productos como: Turbosina, Diésel, coque, combustóleo, gas LP, etc. El producto principal que se genera en el sistema de refinación es la gasolina, la gasolina se ha vuelvo un producto muy importante a nivel mundial, es un recurso no renovable y actualmente también es la principal fuente de energía en los países desarrollados.

La calidad de la gasolina al salir del proceso de refinación es muy importante a nivel comercial, por lo tanto, se busca que los procesos generen gasolina de calidad para tener más ganancias, sin embargo, esto no siempre sucede ya que la calidad del crudo puede ser mala, existen 3 tipos principales de crudo; El crudo pesados, El crudo medio y el crudo Ligero. El bueno rendimiento de la gasolina depende de la calidad del crudo, si es pesado el rendimiento de la gasolina es mala, en cambio si el crudo es ligero tiene buenos rendimientos.

Como sabemos la producción de gasolina de calidad es importante, entonces debemos buscar mejorar los procesos para la generación de esta, es aquí donde podemos mejorar el proceso de desregulación catalítica, en base a las cargas y a las propiedades del proceso, podemos buscar un patrón que nos indique la mejor forma de generar gasolina ligera.

Una forma de hacerlo es por medio de la Minería de datos.

## 2.6 Clasificación

La clasificación es la acción de organizar o situar algo según una determinada característica. La clasificación es un modelo predictivo, en este tipo de modelos se estiman valores de características de interés a partir de valores de otro conjunto de variables, este tipo de modelo aplicado a los datos tiene un gran valor, permite la obtención de conocimiento de grandes cantidades de datos.
En nuestra vida diaria, nuestro cerebro hace clasificación todo el tiempo por ejemplo cuando reconocemos a un amigo o familiar por la calle, el cerebro ya hizo un proceso de clasificación que nos permite identificar a personas conocidas, o incluso en las fábricas industriales en el momento de separar el producto dañado con el no dañado, la clasificación nos ayuda a tener las cosas más organizadas, y así poder sacar provecho de ello.

## 2.7 Clasificadores

Los clasificadores son procesos que tienen por objetivo etiquetar un conjunto de datos llamado conjunto de prueba en base a otro conjunto de datos ya etiquetado llamado conjunto entrenamiento.

Los clasificadores son parte fundamental de la minería de datos, Existen varios tipos de clasificadores, algunos de estos tipos son;

- Clasificadores Probabilísticos.
- Clasificadores Arboles de decisión.
- Clasificadores Basado en casos.
- Clasificadores bioinspirados.
- Clasificadores basados en reglas de decisión.

El proceso para generar un clasificador se puede dividir en 2 partes:

- **Fase de Construcción:** se construye un modelo que en base a ello clasifique las instancias de manera más fácil y eficiente.
- **Fase Clasificación:** se busca clasificar un conjunto de instancias con ayuda del modelo ya construido.

Los datos de entrada se representan por medio de tablas, donde las columnas representan atributo y los registros son llamados Instancias, en el conjunto de entrenamiento la última columna se utiliza para las etiquetas de las instancias.

| Atributo 1 | Atributo 2 | Atributo 3 | Atributo 4 | Atributo 5 | Clase |
|---|---|---|---|---|---|
| Valor $_{11}$ | Valor $_{21}$ | Valor $_{31}$ | Valor $_{41}$ | Valor $_{51}$ | Etiqueta |
| Valor $_{12}$ | Valor $_{22}$ | Valor $_{32}$ | Valor $_{42}$ | Valor $_{52}$ | Etiqueta |
| Valor $_{13}$ | Valor $_{23}$ | Valor $_{33}$ | Valor $_{43}$ | Valor $_{53}$ | Etiqueta |

Tabla 2.10 Representación de datos de entrada.

Los valores pueden ser de distintos tipos de datos, es decir, que el contenido de la base de datos puede ser de tipo numérico o de tipo cadena. En el Conjunto Entrenamiento existen instancias ya clasificadas que nuestro clasificador necesita para construir su modelo. En el conjunto prueba cada ejemplo o instancia debe de ser etiquetada con ayuda de nuestro modelo de clasificación. El clasificador se encargará de decidir que instancias pertenecen a un grupo determinado.
Las instancias se pueden visualizar como eventos, objetos, acciones, etc. Los atributos son características que pertenecen a cada instancia, estas características definen y le dan sentido. Se le llama clases a los diferentes tipos de etiquetas que se pueden agrupar las instancias.

## 2.7.1 Bayes Ingenuo

Bayes ingenuo es un tipo de clasificador probabilístico, en este clasificador se utiliza la probabilidad condicional y la probabilidad a priori. Bayes Ingenuo esta entre los algoritmos más utilizados, se usa cuando el conjunto datos de entrenamiento no es muy grande. Entre las aplicaciones exitosas están los diagnósticos médicos para prever enfermedades y en la clasificación de documentos de texto.

**Teoría**

En general, el clasificador Bayes ingenuo se puede ver como una función, puede parecer algo extraña y difícil:

$$NaiveBayes(a_1, a_2, \dots, a_n) = arg\max_{c_j}^{m} p(C_j) \prod_{i=1}^{n} p(a_i|C_j)$$

Donde

A = $\{a_1, a_2, \dots, a_n\}$ Instancia a clasificar

C = $\{C_1, C_2, \dots, C_m\}$ conjunto de clases

Si desarrollamos el producto se vera de esta forma:

$$NaiveBayes(a_1, a_2, \dots, a_n) = arg\max_{c_j}^{m} p(c_j)\, p(a_1|c_j) p(a_2|c_j) \dots p(a_m|c_j)$$

Ahora bien, si desarrollamos argmax para cada Clase c, tendremos las siguientes igualdades:

$$NaiveBayes(a_1, a_2, \dots, a_n) = argmax_c \begin{cases} p(c_1)p(a_1|c_1)p(a_2|c_1) \dots p(a_n|c_1) \\ p(c_2)p(a_1|c_2)p(a_2|c_2) \dots p(a_n|c_2) \\ \vdots \\ \vdots \\ p(c_m)p(a_1|c_m)p(a_2|c_m) \dots p(a_n|c_m) \end{cases}$$

Podemos darnos cuenta que tendremos tantas líneas de probabilidades como clases "c", hasta ahora simplemente hemos extendido la función de Bayes ingenuo, pero nos puede surgir una pregunta:
¿Qué significa $argmax_c$? argmax es un operador que acepta varios operandos, argmax busca el resultado máximo de sus operandos para tomar su parámetro principal ($C_i$).
Veamos un ejemplo

$$argmax_c \begin{cases} p(C_1)p(a_1|C_1)p(a_2|C_1) \dots p(a_n|C_1 \\ p(C_2)p(a_1|C_2)p(a_2|C_2) \dots p(a_n|C_2 \\ \vdots \\ \vdots \\ p(C_m)p(a_1|C_m)p(a_2|C_m) \dots p(a_n|C \end{cases}$$

Supongamos que se realizaron las operaciones numéricas de cada línea y resulto que la 2da línea es mayor que todas las demás, entonces la mayor fue la línea $p(C_2)p(a_1|C_2)p(a_2|C_2) \dots p(a_n|C_2)$ pero lo que se busca es el parámetro principal del que gano, en este caso es "$C_2$". Por lo tanto: $NaiveBayes(a_1, a_2, \dots, a_n) = C_2$

Quedo claro que el clasificador Bayes Ingenuo debe calcular probabilidades por cada clase que exista, para después buscar el máximo y sacar la $C_i$.
Ahora vamos a justificar toda la formula.

Lo que se busca es la clasificación de la instancia $a_1, a_2, \dots, a_n$ para saber a qué clase pertenece para ello se deben calcular las siguientes probabilidades:

$p(C \mid a_1, a_2, \dots, a_n)$ Probabilidad de que sea de la clase $C_1$ dado que pasa $a_1, a_2, \dots, a_n$

$p(C_2 \mid a_1, a_2, \dots, a_n)$ Probabilidad de que sea de la clase $C_2$ dado que pasa $a_1, a_2, \dots, a_n$

:

$p(C_m \mid a_1, a_2, \dots, a_n)$ Probabilidad de que sea de la clase $C_m$ dado que pasa $a_1, a_2, \dots, a_n$

Si sabemos cuál es más probable, entonces escogemos la Clase ganadora $C_{ganadora}$ .

Podemos ver que cada probabilidad esta condiciona con múltiples $a_1, a_2, \dots, a_n$ , el problema es que si el número *n* de variables independientes es grande (o cuando éstas pueden tomar muchos valores), entonces basar este modelo en tablas de probabilidad se vuelve imposible. Por lo tanto el modelo se reformula para hacerlo más manejable:

Parte del Nombre que se le da al clasificador "Bayes" es porque se usa en este momento el Teorema de Bayes.

Teorema de Bayes

$$p(A_i \mid B) = \frac{p(B|Ai)\, p(A_i)}{p(B)}$$

Donde:

$p(A_i)$ Son las probabilidades a priori
$p(B|Ai)$ Es la probabilidad de B en la hipótesis $A_i$
$p(A_i \mid B)$ Son las probabilidades a posteriori.

El teorema de Bayes nos ayudara a replantear el problema de otro modo, entonces aplicamos el teorema de Bayes para todas las probabilidades de cada clase, quedando de la siguiente forma.

$$p(C_1 \mid a_1, a_2, \dots, a_n) = \frac{p(a_1, a_2, \dots, a_n \mid C_1)\, p(C_1)}{P(a_1, a_2, \dots, a_n)}$$

$$p(C_2 \mid a_1, a_2, \dots, a_n) = \frac{p(a_1, a_2, \dots, a_n \mid C_2)\, p(C_2)}{P(a_1, a_2, \dots, a_n)}$$

:

$$p(C_m \mid a_1, a_2, \dots, a_n) = \frac{p(a_1, a_2, \dots, a_n \mid C_m)\, p(C_m)}{P(a_1, a_2, \dots, a_n)}$$

Ahora es momento de quitar el denominador de toda la probabilidad, si quitamos el denominador nos queda de la siguiente forma:

$$p(C_1 \mid a_1, a_2, \dots, a_n) = p(a_1, a_2, \dots, a_n \mid C_1)\, p(C_1)$$

$$p(C \mid a_1, a_2, \dots, a_n) = p(a_1, a_2, \dots, a_n \mid C_2)\, p(C_2)$$

:

$$p(C_m \mid a_1, a_2, \dots, a_n) = p(a_1, a_2, \dots, a_n \mid C_m)\, p(C_m)$$

¿Por qué quitaríamos el denominador? es muy simple, porque es una constante, el resultado es el mismo si no se la quitamos, a primera vista parece algo extraño quitar el denominado porque parece que afectara de manera muy grave el resultado pero sin embargo no es así, el resultado resulta ser el mismo, ya que el denominador es el mismo para todas las probabilidades, depende solamente de la instancia y no de la clase.
Imaginemos que no se los quitamos y trabajamos con el denominador durante todo cálculo, la última fase es la de buscar el máximo, pero queda claro que para buscar el máximo debemos de comparar cada una de las probabilidades, de la siguiente forma:

$$\frac{p(a_1, a_2, \dots, a_n \mid c_1)\, p(C_1)}{P(a_1, a_2, \dots, a_n)} \equiv \frac{p(a_1, a_2, \dots, a_n \mid c_2)\, p(C_2)}{P(a_1, a_2, \dots, a_n)}$$

Se compara la probabilidad 1 con la 2 para buscar el máximo, pero como vemos es una desigualdad en la cual como los denominadores son los mismos entonces se cancelan entre sí, podemos concluir que *el denominador no influye en el resultado por que no buscamos el valor numérico exacto, sino el máximo*.

Ya simplificada la expresión vamos a transformarla más, los numeradores son equivalente a una probabilidad compuesta.

$$p(c_1 \mid a_1, a_2, \dots, a_n) = p(a_1, a_2, \dots, a_n \mid c_1)\, p(C_1)$$

$$p(c_2 \mid a_1, a_2, \dots, a_n) = p(a_1, a_2, \dots, a_n \mid c_2)\, p(C_2)$$

:

$$p(c_m \mid a_1, a_2, \dots, a_n) = p(a_1, a_2, \dots, a_n \mid c_m)\, p(C_m)$$

Que puede ser reescrita como sigue, aplicando repetidamente la definición de probabilidad condicional:

$$p(c_1 \mid a_1, a_2, \dots, a_n) = p(c_1)p(a_1|c_1)\, p(a_2|c_1, a_1) \dots p(a_n|C_1, a_1, a_2, \dots, a_n)$$
$$p(c_1 \mid a_1, a_2, \dots, a_n) = p(c_2)p(a_1|c_2)\, p(a_2|c_2, a_1) \dots p(a_n|C_2, a_1, a_2, \dots, a_n)$$
$$\vdots$$
$$p(c_1 \mid a_1, a_2, \dots, a_n) = p(c_m)p(a_1|c_m)\, p(a_2|c_m, a_1) \dots p(a_n|C_m, a_1, a_2, \dots, a_n)$$

Ahora es cuando la otra parte del nombre "ingenuo" de independencia condicional entra en juego: se asume que cada $a_i$ es independiente de cualquier otro $a_j$ para $j \neq i$. Esto significa que la probabilidad compuesta puede expresarse como:

$$p(c_1 \mid a_1, a_2, \dots, a_n) = p(c_1)p(a_1|c_1)\, p(a_2|c_1) \dots p(a_n|C_1)$$
$$p(c_1 \mid a_1, a_2, \dots, a_n) = p(c_2)p(a_1|c_2)\, p(a_2|c_2) \dots p(a_n|C_2)$$
$$p(c_1 \mid a_1, a_2, \dots, a_n) = p(c_m)p(a_1|c_m)\, p(a_2|c_m) \dots p(a_n|C_m)$$

Ahora podemos ver que ya llegamos a la función de Bayes ingenuo planteada desde el principio.
Ventajas:

- Es fácil de implementar
- Obtiene buenos resultados en gran parte de los casos

Desventajas:

- Asumir que las variables tienen independencia condicional respecto a la clase lleva a una falta de exactitud.

**¿Cómo funciona?**
Para entender cómo se comporta el clasificador vamos a realizar un ejemplo práctico.

**Contexto del problema**
*Lenin es un tenista que le gusta jugar todas las mañanas en un parque cercano, el registró en una libreta los siguientes atributos de los últimos 14 días; Cielo, Temperatura, Humedad, Viento. Esto con el fin de decidir si saldrá a jugar al parque, ya que en los últimos tres meses el clima no se ha puesto a su favor y se ha sentido mal algunos días, Lenin no sabe si se sentirá mal cuando vaya a jugar, por lo tanto debe de decidir "SI Jugara" o "NO jugara".*

**Objetivo**
Para ayudarle a Lenin vamos a diagnosticar si ese día debe *"SI Jugara"* o *"NO jugara".*

**Conjunto Prueba**
Lenin observa el día, y nos indica los atributos que tiene ese día:

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Fría | Alta | Cierto | ? |

**Tabla 2.11 Instancia prueba.**

*El conjunto prueba es un conjunto de instancias que son tratadas por los clasificadores para etiquetarlas con una clase, no tiene que ser solamente una instancia, por motivos prácticos aquí se planteó solo una instancia.*

**Conjunto Entrenamiento**

Sabemos que Lenin registró los atributos de los últimos 14 días en una libreta con el correspondiente resultado del día, el cual es "SI debe salir" y "No debió salir", es importante notar que estamos en un problema con Aprendizaje supervisado ya que sabemos la clase de cada registro de la libreta, los registros de la libreta se comportan como el conjunto entrenamiento, el conjunto entrenamiento nos ayudara a predecir si hoy debe jugar o no debe jugar.

A continuación, los registros de la libreta en forma de tabla:

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Alta | Alta | Débil | No |
| Soleado | Alta | Alta | Fuerte | No |
| Cubierto | Alta | Alta | Débil | Si |
| Lluvioso | Suave | Alta | Débil | Si |
| Lluvioso | Fría | Normal | Débil | Si |
| Lluvioso | Fría | Normal | Fuerte | No |
| Cubierto | Fría | Normal | Fuerte | Si |
| Soleado | Suave | Alta | Débil | No |
| Soleado | Fría | Normal | Débil | Si |
| Lluvioso | Suave | Normal | Débil | Si |
| Soleado | Suave | Normal | Fuerte | Si |
| Cubierto | Suave | Alta | Fuerte | Si |
| Cubierto | Alta | Normal | Débil | Si |
| Lluvioso | Suave | Alta | Fuerte | No |

Tabla 2.12 Conjunto Entrenamiento

Después de plantearnos el problema e identificar los datos iniciales, vamos a construir el clasificador, para ello se sugiere realizar el conteo de los valores de los atributos.

| Cielo | Temperatura | Humedad | Viento | Lluvia |
|---|---|---|---|---|
| Soleado | Alta | Alta | Débil | No |
| Soleado | Alta | Alta | Fuerte | No |
| Cubierto | Alta | Alta | Débil | Si |
| Lluvioso | Suave | Alta | Débil | Si |
| Lluvioso | Fría | Normal | Débil | Si |
| Lluvioso | Fría | Normal | Fuerte | No |
| Cubierto | Fría | Normal | Fuerte | Si |
| Soleado | Suave | Alta | Débil | No |
| Soleado | Fría | Normal | Débil | Si |
| Lluvioso | Suave | Normal | Débil | Si |
| Soleado | Suave | Normal | Fuerte | Si |
| Cubierto | Suave | Alta | Fuerte | Si |
| Cubierto | Alta | Normal | Débil | Si |
| Lluvioso | Suave | Alta | Fuerte | No |

Para la clase: SI
Atributo: Humedad
Valor: Normal
Contamos las filas que cumplen con estos 3 requisitos.
En este caso el conteo es de 6.

Figura 2.13 conteo de entrenamiento

| Cielo | Temperatura | Humedad | Viento | Lluvia |
|---|---|---|---|---|
| Soleado | Alta | Alta | Débil | No |
| Soleado | Alta | Alta | Fuerte | No |
| Cubierto | Alta | Alta | Débil | Si |
| Lluvioso | Suave | Alta | Débil | Si |
| Lluvioso | Fría | Normal | Débil | Si |
| Lluvioso | Fría | Normal | Fuerte | No |
| Cubierto | Fría | Normal | Fuerte | Si |
| Soleado | Suave | Alta | Débil | No |
| Soleado | Fría | Normal | Débil | Si |
| Lluvioso | Suave | Normal | Débil | Si |
| Soleado | Suave | Normal | Fuerte | Si |
| Cubierto | Suave | Alta | Fuerte | Si |
| Cubierto | Alta | Normal | Débil | Si |
| Lluvioso | Suave | Alta | Fuerte | No |

Para la clase: NO
Atributo: Temperatura
Valor: Fría
Contamos las filas que cumplen con estos 3 requisitos.
En este caso el conteo es de 1.

Tabla 2.14 Conteo de atributo Temperatura y clase NO

Siguiendo con el conteo, se llenara una tabla como la que sigue:

| Clase SI | | Clase NO | |
|---|---|---|---|
| Pronostico | | Pronostico | |
| Soleado | 2 | Soleado | 3 |
| Cubierto | 4 | Cubierto | 0 |
| Lluvioso | 3 | Lluvioso | 2 |
| Temperatura | | Temperatura | |
| Alta | 2 | Alta | 2 |
| Suave | 4 | Suave | 2 |
| Fría | 3 | Fría | 1 |
| Humedad | | Humedad | |
| Alta | 3 | Alta | 4 |
| Normal | 6 | Normal | 1 |
| Viento | | Viento | |
| Fuerte | 3 | Fuerte | 3 |
| Débil | 6 | Débil | 2 |
| Lluvia | | Lluvia | |
| Si | 9 | No | 5 |

**Tabla 2.15 Estructura para guardar conteo**

La tabla muestra todo el conteo que se realizó.

El conteo de cada valor de cada atributo de su respectiva clase será muy importante ya que Bayes ingenuo es un Clasificador probabilístico y como sabemos la probabilidad clásica se maneja de la siguiente forma:

$$P(A) = \frac{Numero\ de\ casos\ del\ eventos\ A}{Numero\ total\ de\ eventos}$$

Entonces el conteo servirá para calcular el numerador o el denominador de la probabilidad correspondiente.

*Se recomienda realizar este tipo de tabla cuando el conjunto prueba es grande, ya que puede resultar muy útil a la hora de calcular las probabilidades, en este caso el conjunto prueba solo es de una instancia, sin embargo, no deja de ser útil como muestra.*

Ahora como tenemos 2 tipos de clases "SI" y "NO", La fórmula indica que tenemos que calcular dos probabilidades una para cada clase, al último vamos a comparar dichas probabilidades para ver cuál es la más probable y poder decidir la clase correspondiente a la instancia del conjunto de prueba.

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Fría | Alta | Fuerte | ? |

Tabla 2.16 Instancia prueba a clasificar

$$P(si|soleado, fria, alta, fuerte) = p(si)p(soleado|si)p(fria|si)p(alta|si)p(fuerte|si)$$

$$P(no|soleado, fria, alta, fuerte) = p(no)p(soleado|no)p(fria|no)p(alta|no)p(fuerte|no)$$

Gracias a la tabla en donde tenemos el conteo y la fórmula de probabilidad clásica, podemos sacar fácilmente las probabilidades.

Por ejemplo calculemos algunas probabilidades.

$$p(si) = \frac{Numero\ filas\ con\ clase\ SI}{Numero\ total\ de\ filas} = \frac{9}{14} = 0.6428$$

$$p(soleado|si) = \frac{Numero\ de\ filas\ con\ soleado\ y\ SI}{numero\ de\ filas\ con\ SI} = \frac{2}{9} = 0.2222$$

$$p(no) = \frac{Numero\ filas\ con\ clase\ NO}{Numero\ total\ de\ filas} = \frac{5}{14} = 0.3571$$

Sustituyendo los valores numéricos correspondientes nos dará la siguiente expresión:

$$P(si|soleado, fria, alta, fuerte) = \left(\frac{9}{14}\right)\left(\frac{2}{9}\right)\left(\frac{3}{9}\right)\left(\frac{3}{9}\right)\left(\frac{3}{9}\right) = 0.0053$$

$$P(no|soleado, fria, alta, fuerte) = \left(\frac{5}{14}\right)\left(\frac{3}{5}\right)\left(\frac{1}{5}\right)\left(\frac{4}{5}\right)\left(\frac{3}{5}\right) = 0.0206$$

Ya calculadas las probabilidades vamos a comprobar cuál es la más probable.

$P(si|\ldots) < P(no|\ldots) \equiv 0.0053 < 0.0206$ Por lo tanto, La clase que gana es NO.

Como la clase más probable es "NO" entonces la instancia que estamos clasificando tendrá el diagnostico "NO", Lenin no debe ir a jugar este día.

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Fría | Alta | Fuerte | ? |

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Fría | Alta | Fuerte | NO |

Tabla 2.17 Transición de clasificación.

*Todo este proceso se debe de realizar para cada una de las instancias del conjunto prueba, con el fin de etiquetarlas con una clase.*

## 2.7.2 K-vecinos más cercanos

**Descripción**

K- vecinos más cercanos es uno de los procesos de clasificación más simples y fácil de implementar con respecto a otros clasificadores, este tipo de clasificadores se basan en las cercanías de los datos dentro del plano, la forma en la que representa es por medio de coordenadas, lo que se busca es clasificar una instancia en forma de coordenada en el plano y buscar sus vecinos más cercanos para realizar un criterio de votación y decidir qué tipo de clase pertenece, el criterio más sencillo es la mayoría de clases.

Como tal K-vecinos más cercano no construye un modelo, todo el algoritmo y proceso se realiza para cada instancia del conjunto prueba, podemos decir que en el mismo proceso de construcción también se clasifica.

$$KNN(a_1, a_2, \dots, a_n) = concenso(\ los\ vecinos\ mas\ cercanos, a_1, a_2, \dots, a_n)$$

Para representar los puntos en el plano de n-dimensiones, se utiliza las tuplas $(a_1, a_2, \dots, a_n)$.

Para medir la distancia de puntos en el plano lo más común es utilizar la distancia euclidiana, hay que notar que para representar los datos en un plano de n dimensiones se necesita de elementos numéricos, se podría pensar que esta condición limita solamente a base de datos numéricos, sin embargo se puede transformar cualquier bases de datos a una base de datos numérica.
Los criterios para la trasformación son variados pero la conversión de String a numérica por medio del código ASCII es buena opción.
El algoritmo que describe el proceso de clasificación es el siguiente:

Dada la instancia prueba: $(a_1, a_2, \dots, a_n)$, el conjunto entrenamiento también representada en tuplas y k= número de vecinos a considerar para el criterio de votacion.

1- Representar el conjunto prueba y el conjunto entrenamiento en el plano
2- calcular la distancia entre la instancia prueba $(a_1, a_2, \dots, a_{n-1})$ y cada tupla del conjunto entrenamiento, La fórmula es la siguiente:
   Si el conjunto entrenamiento es el siguiente:
   $(e_{11}, e_{12}, \dots, e_{1n})$

$(e_{21}, e_{22}, \dots, e_{2n})$
$\vdots$
$(e_{m1}, e_{m2}, \dots, e_{mn})$

$$Distancia_i = \sqrt{(e_{i1} - a_1)^2 + (e_{i2} - a_2)^2 + \cdots + (e_{i(n-1)} - a_{n-1})^2}$$

3- Se buscan las k distancias más cercanas y realizan un consenso por mayoría de votos donde cada vecino propone su clase como etiqueta para la instancia prueba, es recomendable que el numero k vecinos sea un número impar, para que no haya empates en el momento que se vote.
4- Con la clase que obtuvo más votos se etiqueta la instancia prueba.

**¿Cómo funciona?**

Para entender cómo se comporta el clasificador vamos a realizar un ejemplo práctico.

**Datos**

K Vecinos

La constante K considerado para este ejemplo será de 3.

**Conjunto entrenamiento**

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 |
| 2 | 2 | 2 | 1 | 2 |
| 1 | 2 | 2 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 |

**Tabla 2.18 conjunto entrenamiento para K vecinos más cercanos**

*El conjunto entrenamiento ya fue convertido a datos numéricos.*

**Conjunto prueba**

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| 3 | 2 | 1 | 2 | ? |

**Tabla 2.19 conjunto prueba para K vecinos más cercanos**

**Clasificación**

Siguiente el algoritmo vamos a representar los datos como coordenadas

Entrenamiento

E1 (2,2,2,2,2)
E2 (2,2,2,1,2)
E3 (1,2,2,1,1)
E4 (3,1,1,1,1)

Prueba

P1 (3,2,1,2)

Ahora vamos a calcular las distancias:

E1-P1 $\sqrt{(2-3)^2 + (2-2)^2 + (2-1)^2 + (2-2)^2} = 1.4142$

E2-P1 $\sqrt{(2-3)^2 + (2-2)^2 + (2-1)^2 + (1-2)^2} = 1.7320$

E3-P1 $\sqrt{(1-3)^2+(2-2)^2+(2-1)^2+(1-2)^2} = 4.4494$

E4-P1 $\sqrt{(3-3)^2+(1-2)^2+(1-1)^2+(1-2)^2} = 1.4142$

Ya calculadas las distancias vamos a tomar las primeras k distancias más pequeñas, estas son:

E1 (2,2,2,2,2)
E4 (3,1,1,1,1)
E2 (2,2,2,1,2)

Ahora vamos por el criterio de votación de mayoría de clase, Gana la clase "2".
Por tanto la instancia prueba queda de la siguiente forma

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| 3 | 2 | 1 | 2 | ? = 2 |

**Tabla 2.20 conjunto prueba ya clasificada**

Como vimos resulto muy sencillo realizar las operaciones, sin embargo tenemos que considerar lo siguiente: *En K-vecinos más cercano, es pesado el número de operaciones para calcular la distancia. Ya que se tienen que calcular la distancia de todo el conjunto prueba con respecto a todo el conjunto entrenamiento. Esto indica que el número de operación para calcular las distancias está dada por:*

$$num_{operaciones} = num_{prueba}\ x\ num_{entrenamiento}$$

Para el ejercicio anterior el número de operaciones es:

$$num_{operaciones} = 1\ x\ 4 = 4$$

## 2.7.3 Iterative Dichotomiser 3

El clasificador Iterative Dichotomiser 3 ID3 es un algoritmo basado en arboles de decisión, fue planteado por Ross Quinlan, investigador de ciencias de la computación en minería de datos y teoría de decisión. Este algoritmo busca obtener un mínimo árbol de decisión para representar un conjunto de entrenamiento para clasificar un conjunto prueba.

El modelo ID3 se puede ver de manera gráfica, donde cada nodo representa los diferentes atributos presentes en los datos, las ramificaciones del árbol representan los cambios posibles a seguir.

Para predecir la clase de un nuevo ejemplo, los nodos terminales u hojas establecen la clase a la que pertenece el nuevo ejemplo de prueba si se sigue la ramificación en cuestión, la principal ventaja de usar este tipo de estrategias es que el costo computacional es bastante reducido.

Cada atributo se deberá evaluar para decidir si se incluye en el árbol, este algoritmo se empieza de una árbol vacío y se va llenando de forma recursiva, tomando en cada nodo aquel atributo que tiene el mayor número de información, cada vez que hagamos la elección de un

atributo, debería hacer que los subconjuntos de ejemplos que genera el atributo sean mayoritariamente de una clase.
Para medir esto necesitamos una medida de la cantidad de información que cubre un atributo, a esto se le conoce como medida de entropía.
La primera cosa que debemos formular es la cantidad de información que tiene cada clase la cual se puede hacer de la siguiente forma:

Cantidad de información

Dado un conjunto de ejemplos X clasificados en un conjunto de clases $C = \{ c_1, c_2, \dots, c_n \}$ siendo $|c_i|$ la cardinalidad de la clase $c_i$ y $|X|$ el número total de ejemplo, la cantidad de información vendrá expresada de la siguiente forma:

$$I(X, C) = -\sum_{c_i \in C} \frac{|c_i|}{|x|} \log_2 \left(\frac{|c_i|}{|x|}\right) \qquad (1)$$

Donde podemos notar que el argumento del logaritmo hace referencia a la probabilidad a priori.
Teniendo la cantidad de información se puede definir la Entropía, la Entropía es una medida que nos permite medir que tanto aporte de información nos da algún atributo, puede formular de la siguiente forma:

Entropía

Para cada uno de los atributos $A_i$; siendo $\{v_{i1}, v_{i1}, \dots, v_{in}\}$ el conjunto de posibles valores del atributo $A_i$, $|[A_i (c) = v_{ij}]|$ el numero de ejemplos que tiene el valor $v_{ij}$ en su atributo $A_i$, la función de entropía es expresada de la siguiente forma:

$$E(X, C, A_i) = -\sum_{v_{ij} \in A_i} \frac{|[\mathrm{Ai}\ (\mathrm{c}) = \mathrm{vij}]|}{|x|} I(|[\mathrm{Ai}\ (\mathrm{c}) = \mathrm{vij}]|, C) \qquad (2)$$

Con el resultado de estas medidas se define la función de ganancia de información de la siguiente forma:

$$G(|x|, C, A_i) = I(X, C) + E(X, C, A_i)$$

La ganancia de información es la diferencia entre la cantidad de información que se necesita para hacer una clasificación.

Es por eso que se toma el atributo que maximice este valor para ir expandiendo el árbol de inducción. El pseudocodigo del algoritmo ID3 para la construcción de un árbol de decisión es la siguiente:

Algoritmo ID3

**func** ID3(X;C;A) _ ( X: Ejemplos, C: Clasificación, A: Atributos)

**si** todos los ejemplos son de la misma clase **entonces**

ID3 hoja con la clase.

**otro**

Calcular la función de cantidad de información de los ejemplos (I)

**para** cada atributo en A

Calcular la función de entropía (E) y la ganancia de información (G)

Escoger el atributo que maximiza (G) (sea a)

Eliminar a de la lista de atributos (A)

**termina**

**para** cada partición generada por los valores vi del atributo a

Arboli ID3(ejemplos de X con a = vi,

Clasificación de los ejemplos, Atributos restantes)

Generar árbol con a = viy Arboli

**termina**

ID3 la unión de todos los árboles

**Termina**

A continuación se realizara un ejemplo práctico, con el objetivo de aclarar las ideas anteriores.

Se ocupara la misma base de datos que en la sección 2.7.1 de Bayes Ingenuo

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Alta | Alta | Débil | No |
| Soleado | Alta | Alta | Fuerte | No |
| Cubierto | Alta | Alta | Débil | Si |
| Lluvioso | Suave | Alta | Débil | Si |
| Lluvioso | Fría | Normal | Débil | Si |
| Lluvioso | Fría | Normal | Fuerte | No |
| Cubierto | Fría | Normal | Fuerte | Si |
| Soleado | Suave | Alta | Débil | No |
| Soleado | Fría | Normal | Débil | Si |
| Lluvioso | Suave | Normal | Débil | Si |
| Soleado | Suave | Normal | Fuerte | Si |
| Cubierto | Suave | Alta | Fuerte | Si |
| Cubierto | Alta | Normal | Débil | Si |
| Lluvioso | Suave | Alta | Fuerte | No |

**Tabla 2.21 Conjunto entrenamiento para ID3**

Lo que busca es generar el árbol de decisión.

Calculamos la Cantidad de Información.

|X| = 14
$|C_1|$ = Si = 9
$|C_2|$ = No = 5

Donde:

|X| Es el número de registros en el entrenamiento.
$|C_1|$ Es el número de ejemplos con la clase Si
$|C_2|$ Número de ejemplos con la clase No

$$I(X,C) = -\sum_{c_i \in C} \frac{|c_i|}{|x|} \log_2 \left( \frac{|c_i|}{|x|} \right) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.9402$$

Calculamos la Entropía para cada atributo

$$E(X,C,A_i) = -\sum_{v_{ij} \in A_i} \frac{|[\text{Ai (c)} = \text{vij}]|}{|x|} I(|[\text{Ai (c)} = \text{vij}]|, C)$$

Se recomienda hacer una pequeña tabla con el conteo de los valores del atributo A con cada clase:

| Clase SI | | Clase NO | |
|---|---|---|---|
| Cielo | | Cielo | |
| Soleado | 2 | Soleado | 3 |
| Cubierto | 4 | Cubierto | 0 |
| Lluvioso | 3 | Lluvioso | 2 |
| Temperatura | | Temperatura | |
| Alta | 2 | Alta | 2 |
| Suave | 4 | Suave | 2 |
| Fría | 3 | Fría | 1 |
| Humedad | | Humedad | |
| Alta | 3 | Alta | 4 |
| Normal | 6 | Normal | 1 |
| Viento | | Viento | |
| Fuerte | 3 | Fuerte | 3 |
| Débil | 6 | Débil | 2 |

**Tabla 2.22 Conteo del conjunto entrenamiento**

- Atributo **Cielo**

$$E(X, C, Cielo) = (soleado)\frac{5}{14}\left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + (cubierto)\frac{4}{14}\left(-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}\right) + (lluvioso)\frac{5}{14}\left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) = 0.3467 + 0 + 0.3467 = 0.6935$$

- Atributo **Temperatura**

$$E(X, C, Temperatura) = (alta)\frac{4}{14}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}\right) + (suave)\frac{6}{14}\left(-\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6}\right) + (fria)\frac{4}{14}\left(-\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4}\right) = 0.2857 + 0.3935 + 0.2317 = 0.9109$$

- Atributo **Humedad**

$$E(X, C, Humedad) = (alta)\frac{7}{14}\left(-\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7}\right) + (normal)\frac{7}{14}\left(-\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7}\right) = 0.4926 + 0.2958 = 0.7884$$

- Atributo **Viento**

$$E(X, C, Viento)$$
$$= (fuerte)\frac{6}{14}\left(-\frac{3}{6}\log_2\frac{3}{6}-\frac{3}{6}\log_2\frac{3}{6}\right)$$
$$+ (debil)\frac{8}{14}\left(-\frac{6}{8}\log_2\frac{6}{8}-\frac{2}{8}\log_2\frac{2}{8}\right) = 0.4285 + 0.4635$$
$$= 0.892$$

Calculamos la Ganancia de cada atributo

$$G(|x|, C, A_i) = I(X, C) - E(X, C, A_i)$$

- Atributo **Cielo**

$$G(|x|, C, \text{Cielo}) = 0.9402 - 0.6935 = 0.2467$$

- Atributo **Temperatura**

$$G(|x|, C, \text{Temperatura}) = 0.9402 - 0.9109 = 0.0293$$

- Atributo **Humedad**

$$G(|x|, C, \text{Humedad}) = 0.9402 - 0.7884 = 0.1518$$

- Atributo **Viento**

$$G(|x|, C, \text{Viento}) = 0.9402 - 0.892 = 0.0482$$

El atributo que tiene mayor ganancia es **Cielo**, por lo cual este atributo se convierte en la raíz del árbol y sus hijos sus valores.

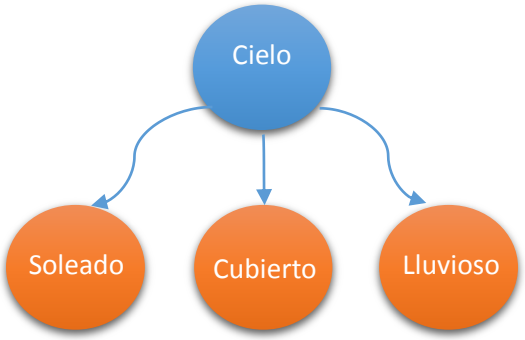



**Figura 2.4 Árbol de decisión**

Ya puesto un nodo se crean las particiones, las particiones consisten en eliminar el atributo cielo de la tabla, y dividir la tabla en los diferentes valores del atributo cielo.

El número de particiones está dada por el número de valores del atributo con mayor ganancia, en este caso **Cielo** tiene 3 particiones**.**

A continuación se muestran las tres particiones generadas:

**Tabla 2.23 Particiones del Atributo Cielo**

| Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|
| Alta | Alta | Débil | No |
| Alta | Alta | Fuerte | No |
| Suave | Alta | Fuerte | No |
| Fría | Normal | Fuerte | Si |
| Suave | Normal | Cierto | Si |

**a) Para el valor Soleado**

| Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|
| Alta | Alta | Fuerte | Si |
| Fría | Normal | Cierto | Si |
| Suave | Alta | Cierto | Si |
| Alta | Normal | Fuerte | Si |

**b) Para el valor cubierto**

| Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|
| Suave | Alta | Fuerte | Si |
| Fría | Normal | Fuerte | Si |
| Fría | Normal | Fuerte | No |
| Suave | Normal | Débil | Si |
| Suave | Alta | Fuerte | No |

**c) Para el valor lluvioso**

Ahora de manera recursiva vamos a tratar cada una de las particiones como otra base de entrenamiento, realizando todo el proceso y los cálculos por cada partición.

| Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|
| Alta | Alta | Débil | No |
| Alta | Alta | Fuerte | No |
| Suave | Alta | Débil | No |
| Fría | Normal | Débil | Si |
| Suave | Normal | Fuerte | Si |

**Tabla 2.24 Partición para el valor soleado**

Se busca Generar el subárbol de decisión para el valor Soleado. Calculamos la Cantidad de Información.

|X| = 5
$|C_1|$ = Si = 2
$|C_2|$ = No = 3

Donde:

|X| Es el número de registros en el entrenamiento.
$|C_1|$ Es el número de ejemplos con la clase Si
$|C_2|$ Número de ejemplos con la clase No

$$I(X,C) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.9709 \qquad (2)$$

Se recomienda hacer una pequeña tabla con el conteo de los valores del atributo A con cada clase:

| Clase SI | | Clase NO | |
|---|---|---|---|
| Temperatura | | Temperatura | |
| Alta | 0 | Alta | 2 |
| Suave | 1 | Suave | 1 |
| Fría | 1 | Fría | 0 |
| Humedad | | Humedad | |
| Alta | 0 | Alta | 3 |
| Normal | 2 | Normal | 0 |
| Viento | | Viento | |
| Fuerte | 1 | Fuerte | 1 |
| Débil | 1 | Débil | 2 |

**Tabla 2.25 Estructura con conteo para sacar la entropía.**

Calculamos la Entropía para cada atributo

Con la formula **(2)**

- Atributo **Temperatura**

$$E(X, C, Temperatura) = (alta)\frac{2}{5}\left(-\frac{0}{2}\log_2\frac{0}{2}-\frac{2}{2}\log_2\frac{2}{2}\right) + (suave)\frac{2}{5}\left(-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}\right) + (fria)\frac{1}{5}\left(-\frac{1}{1}\log_2\frac{1}{1}-\frac{0}{1}\log_2\frac{0}{1}\right) = 0 + 0.4 + 0 = 0.4$$

- Atributo **Humedad**

$$E(X, C, Humedad) = (alta)\frac{3}{5}\left(-\frac{0}{3}\log_2\frac{0}{3}-\frac{3}{3}\log_2\frac{3}{3}\right) + (normal)\frac{2}{5}\left(-\frac{2}{2}\log_2\frac{2}{2}-\frac{0}{2}\log_2\frac{0}{2}\right) = 0 + 0 = 0$$

- Atributo **Viento**

$$E(X, C, Viento) = (\text{Fuerte})\frac{2}{5}\left(-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}\right) + (Debil)\frac{3}{5}\left(-\frac{1}{3}\log_2\frac{1}{3}-\frac{2}{3}\log_2\frac{2}{3}\right) = 0.4 + 0.5509 = 0.9509$$

Calculamos la Ganancia de información de cada atributo

$$G(|x|, C, A_i) = I(X, C) - E(X, C, A_i)$$

- Atributo **Temperatura**

$$G(|x|, C, \text{Temperatura}) = 0.9709 - 0.4 = 0.5709$$

- Atributo **Humedad**

$$G(|x|, C, \text{Humedad}) = 0.9709 - 0 = 0.9709$$

- Atributo **Viento**

$$G(|x|, C, \text{Viento}) = 0.9709 - 0.9509 = 0.02$$

El atributo que tiene mayor ganancia es **Humedad**, por lo cual este atributo se convierte en el nodo y sus ramas sus valores, como estamos realizando la partición soleado, se coloca en ese nodo.

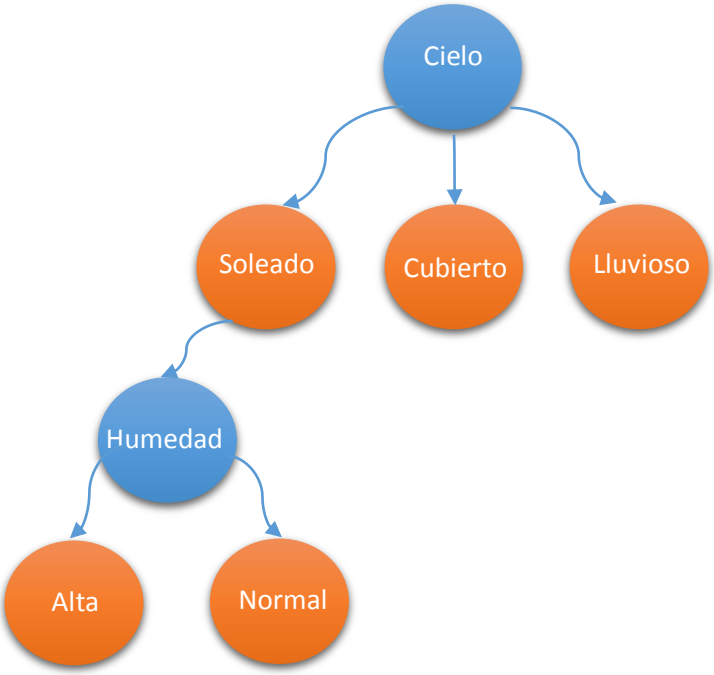



**Figura 2.5 Árbol de decisión con hijo humedad.**

Nuevamente se tienen que crear las particiones, el número de particiones del atributo Humedad es 2.

Siguiendo este algoritmo de manera recursiva, el árbol final es el que sigue.

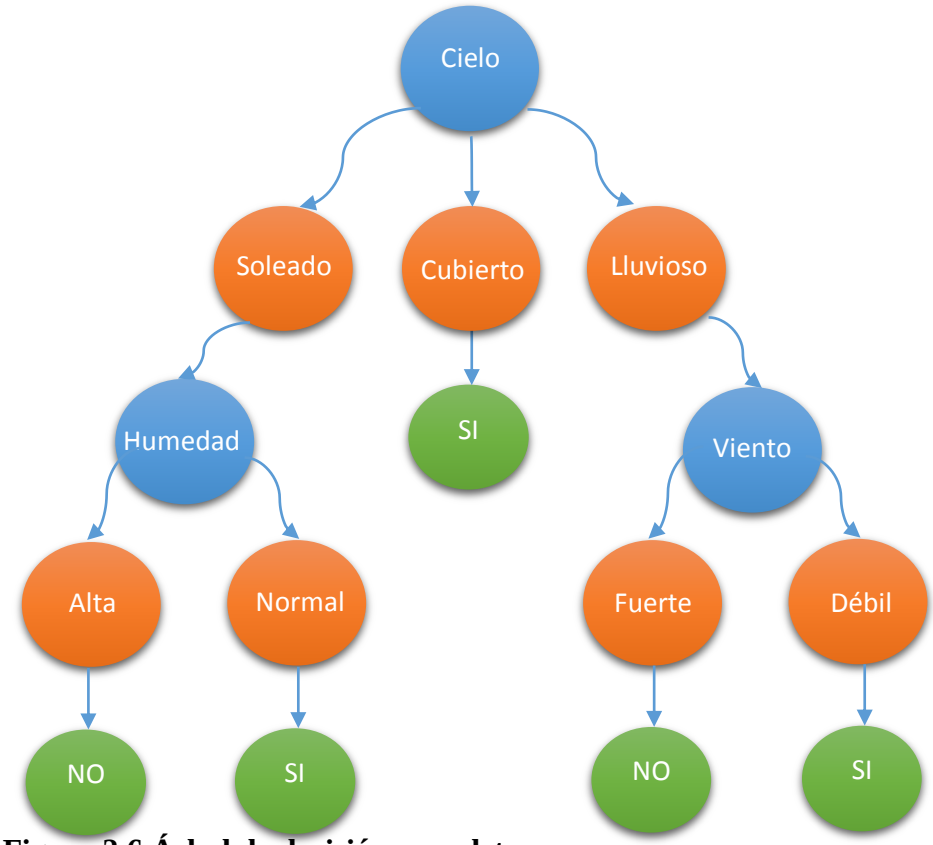



**Figura 2.6 Árbol de decisión completo**

El resultado es un árbol de decisión, este árbol nos servirá para para clasificar instancias de prueba siguiendo las ramas dependiendo de sus atributos.

**Fase de clasificación.**

Se clasificara la siguiente instancia

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Fría | Alta | Fuerte | ? |

Siguiendo las ramas del árbol, nos dará el siguiente resultado.

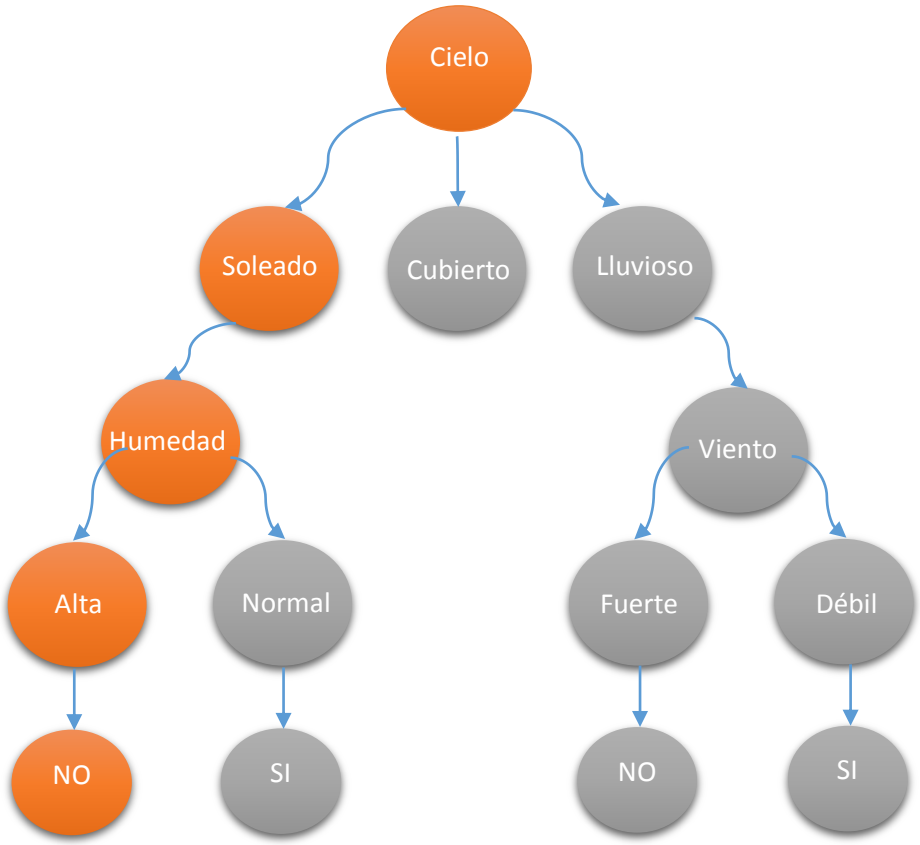



**Figura 2.7 Recorrido de una Clasificación**

Por lo tanto el resultado de la clasificación es

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Fría | Alta | Fuerte | No |

De igual forma se haría este procedimiento para cada rama del árbol.

Ya hemos visto tres clasificadores diferentes, pero necesitamos de una medida de funcionamiento para poder evaluar que tan bien se comportan estos modelos de clasificación. Para poder evaluar el desempeño de los clasificadores existen las medidas de funcionamiento, las cuales se hablarán de ellas en la siguiente sección.

# 2.8 Medidas de Funcionamiento

Para poder evaluar el desempeño de los clasificadores existen diferentes medidas de funcionamiento como son: exactitud (accuracy), precisión (precision), memoria (recall) y mejora (litf).

Utilizaremos la medida de funcionamiento exactitud para evaluar los tres clasificadores que se implementaron en este proyecto terminal.

La medida de funcionamiento exactitud se define de la siguiente forma:

$$exactitud = \frac{(a+d)}{(a+b+c+d)}$$

Donde:

$(a, d)$ son los elementos correctamente clasificados

$(b, c)$ son los elementos incorrectamente clasificados

Con la exactitud sabremos cuantos ejemplos del conjunto prueba fueron clasificados de manera correcta, es decir recibieron la clase que les corresponde del conjunto total de ejemplos.

Para comprender bien este concepto se mostrará un pequeño ejemplo que ilustre la medida de funcionamiento exactitud.

Como se mencionó para poder realizar una clasificación es necesario un conjunto de entrenamiento y un conjunto prueba, para este ejemplo se presenta el conjunto entrenamiento en la **Tabla 2.10** y el conjunto prueba en la **Tabla 2.11.** Estos conjuntos cuentan con 4 atributos que son: Cielo, Temperatura, Humedad y Viento, 14 ejemplos y la clase puede ser de dos tipos: No y Si.

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Alta | Alta | Débil | No |
| Soleado | Alta | Alta | Fuerte | No |
| Cubierto | Alta | Alta | Débil | Si |
| Lluvioso | Suave | Alta | Débil | Si |
| Lluvioso | Fría | Normal | Débil | Si |
| Lluvioso | Fría | Normal | Fuerte | No |
| Cubierto | Fría | Normal | Fuerte | Si |
| Soleado | Suave | Alta | Débil | No |
| Soleado | Fría | Normal | Débil | Si |
| Lluvioso | Suave | Normal | Débil | Si |
| Soleado | Suave | Normal | Fuerte | Si |
| Cubierto | Suave | Alta | Fuerte | Si |
| Cubierto | Alta | Normal | Débil | Si |
| Lluvioso | Suave | Alta | Fuerte | No |

Tabla 2.10 Conjunto entrenamiento “Jugar Tenis”

| Cielo | Temperatura | Humedad | Viento |
|---|---|---|---|
| Soleado | Alta | Alta | Débil |
| Soleado | Alta | Alta | Fuerte |
| Cubierto | Alta | Alta | Débil |
| Lluvioso | Suave | Alta | Débil |
| Lluvioso | Fría | Normal | Débil |
| Lluvioso | Fría | Normal | Fuerte |
| Cubierto | Fría | Normal | Fuerte |
| Soleado | Suave | Alta | Débil |
| Soleado | Fría | Normal | Débil |
| Lluvioso | Suave | Normal | Débil |
| Soleado | Suave | Normal | Fuerte |
| Cubierto | Suave | Alta | Fuerte |
| Cubierto | Alta | Normal | Débil |
| Lluvioso | Suave | Alta | Fuerte |

Tabla 2.11 Conjunto prueba "Jugar Tenis"

Supongamos que después de la clasificación, el conjunto prueba se clasifica de la siguiente forma como se muestra en la **Tabla 2.12.**

| Cielo | Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|---|
| Soleado | Alta | Alta | Débil | Si |
| Soleado | Alta | Alta | Fuerte | No |
| Cubierto | Alta | Alta | Débil | Si |
| Lluvioso | Suave | Alta | Débil | No |
| Lluvioso | Fría | Normal | Débil | Si |
| Lluvioso | Fría | Normal | Fuerte | No |
| Cubierto | Fría | Normal | Fuerte | Si |
| Soleado | Suave | Alta | Débil | Si |
| Soleado | Fría | Normal | Débil | Si |
| Lluvioso | Suave | Normal | Débil | Si |
| Soleado | Suave | Normal | Fuerte | Si |
| Cubierto | Suave | Alta | Fuerte | Si |
| Cubierto | Alta | Normal | Débil | Si |
| Lluvioso | Suave | Alta | Fuerte | Si |

Tabla 2.12 Conjunto prueba "Jugar Tenis" clasificado

Si comparamos las dos columnas de las clases del conjunto entrenamiento y el conjunto prueba, observamos que no todos los elementos fueron clasificados correctamente como se muestra en la Figura 2.13.

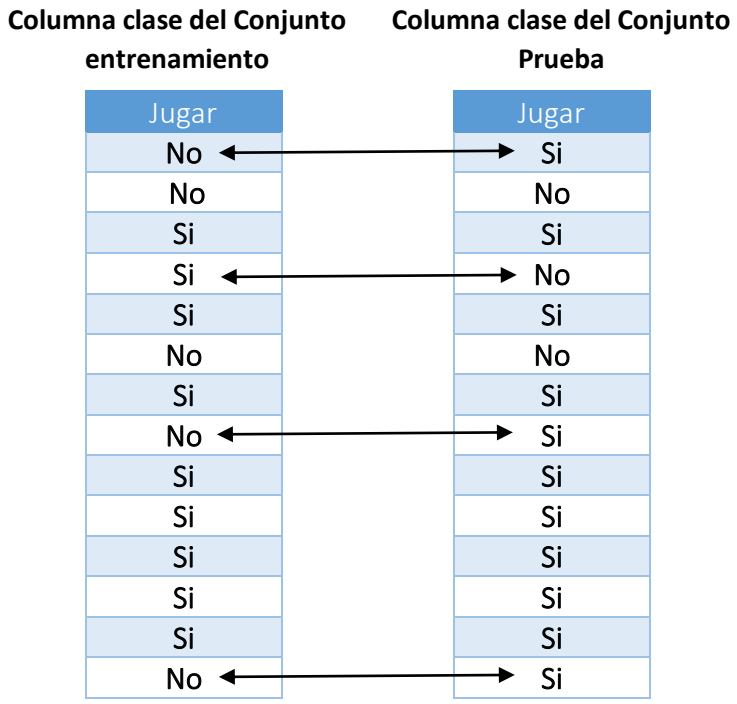



Figura 2.13 Comparando clases para sacar la exactitud

Como no fueron clasificados correctamente todos los elementos, la medida de funcionamiento exactitud no puede ser del 100%.

Para calcular la exactitud necesitamos de:

- $(a, d)$ Los elementos correctamente clasificados = 10
- $(b, c)$ Los elementos incorrectamente clasificados = 4

Obteniendo una exactitud de:

$$exactitud = \frac{10}{14} = 0.71 \; o \; 71\%$$

Por lo cual se tiene del conjunto entrenamiento un 29% de los ejemplos mal clasificados, la cual se denomina la tasa de error del proceso de clasificación.

# Minería de Datos para el análisis de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico

## 3.1. Introducción

Con ayuda de la parte teórica de la sección 2, se llevó a cabo una aplicación que nos ayudó a realizar experimentos con los repositorios de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico.

La forma en la que se abordó la implementación se detallara en esta sección, se expondrán las estrategias de programación que ayudaron a realizar los objetivos, así como resolver los problemas que fueron surgiendo, también se explicaran los detalles más importantes a nivel de código con el fin de recalcar como la teoría se volvió en práctica.

**Características**

Java fue el lenguaje de programación que se consideró para la construcción de la aplicación, actualmente Java es uno de los lenguajes de programación más utilizados en el campo laboral, científico y educativo, algunas de las características y ventajas que se tomaron en consideración para esta decisión fueron las siguientes;

I. El lenguaje Java permite crear cualquier tipo de aplicación, ya sea de escritorio, dispositivo móvil o web, esto permite tener cierta flexibilidad al tener la posibilidad de cambiar de dispositivos sin cambiar mucho el lenguaje de programación.
II. Java es un lenguaje Orientado a Objetos, este concepto ayuda a construir los componentes de la aplicación en objetos haciendo que la interpretación del código sea más fácil, además de aprovechar todas las ventajas que posee la programación orientado a objetos – POO, como Herencia y Polimorfismo.
III. Java cuenta con múltiples librerías que permiten el desarrollo de aplicaciones completas, cuenta con librerías para crear interfaces graficas de usuario ( Swing ) o para crear gráficos ( JFreeChart ) de manera fácil y práctica.
IV. Java cuenta con una gran variedad de Entornos de Desarrollo Integrados – IDE, software que permite el desarrollo de aplicaciones, algunos de ellos permiten corregir errores desde línea de código esto concede una ventaja al programador.

El desarrollo de la implementación se dividió en 3 partes:

1. Parámetros Iniciales.
2. Descripción de la funcionalidad.
3. Descripción a nivel código.

**Parámetros Iniciales**

Comenzar analizando las herramientas que nos serán de utilidad es necesario para evitar problemas a lo largo de la construcción de la aplicación.

La creación de aplicaciones en Java necesita herramientas de desarrollo como Java Development Kit – JDK. JDK es una herramienta que incluye componentes importantes y necesarios como Java Runtime Environment, el compilador Java y las API de Java. La versión utilizada para el desarrollo de la aplicación fue JDK 7 Edición Estándar.
El IDE utilizado para el desarrollo de la aplicación fue Eclipse Jee Mars, Eclipse es un entorno de desarrollo de software compuesta por herramientas de programación, una característica muy fuerte de eclipse es que dispone de un editor de textos con un analizador sintáctico, y la compilación es en tiempo real.

# 3.2. Funcionalidad de la aplicación

En esta sección se detallara el funcionamiento de la aplicación, las características principales e interpretación de los resultados arrojados por la aplicación. La Figura 3.1 muestra la ventana principal de la aplicación, esta surge al inicio de la ejecución.

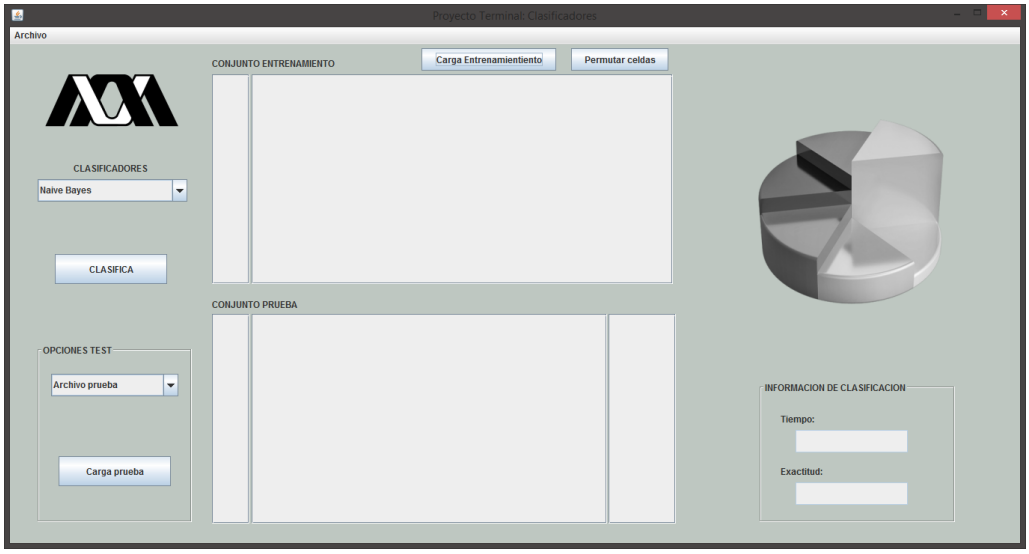



**Figura 3.1 Ventana principal**

A continuación se explica cada componente de la ventana principal:

I. **Carga Conjunto entrenamiento:** La aplicación cuenta con un botón para introducir el conjunto entrenamiento necesario para la clasificación del conjunto prueba.

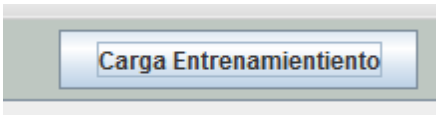



**Figura 3.2 Boton Carga Entrenamiento**

En la aplicación el conjunto entrenamiento se representa con un archivo txt, como se vio en la teoria en la seccion 2 el conjunto entrenamiento y el conjunto prueba se puede visualizar por medio de tablas, por lo cual los archivo tienen la siguiente estructura:

| Pronostico Temperatura Humedad Viento Clase |
|---|
| soleado calor alta debil no |
| soleado calor alta fuerte no |
| nublado calor alta debil si |
| lluvia templado alta debil si |
| lluvia frio normal debil si |
| lluvia frio normal fuerte no |
| nublado frio normal fuerte si |
| soleado templado alta debil no |
| soleado frio normal debil si |
| lluvia templado normal debil si |
| soleado templado normal fuerte si |
| nublado templado alta fuerte si |
| nublado calor normal debil si |
| lluvia templado alta fuerte no |
| nublado calor normal fuerte si |

**Figura 3.3 Estructura de los conjuntos entrenamiento y prueba**

Como se puede observar la estructura del conjunto entrenamiento se represento por medio de una tabla, donde la primera fila son los titulos de los atributos y las filas restantes son los valores de dichos atributos, la separacion de las columnas fue por medio de un espacio en blanco y la ultima columna es la clase a la que pertence toda la fila.
Estos archivos txt son cargados en la aplicación para utilizarlos ya sea para entrenamiento o para prueba.

II. **Distribución del conjunto entrenamiento:** En la ventana se muestra una pequeña sección en la que aparecerá la distribución de las clases del conjunto entrenamiento por medio de una gráfica en forma de pastel (Véase la figura 3.4).

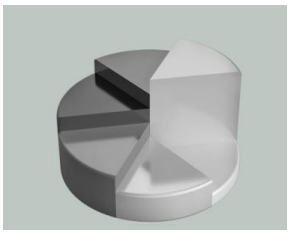


Figura 3.4 Grafica de pastel

Al principio, la aplicación cuenta con una imagen en blanco y negro de una gráfica de pastel, con el fin de mostrar al usuario la parte en donde se visualizara la distribución del conjunto entrenamiento, un ejemplo de la distribución de una clase véase la Figura 3.4.1

Un ejemplo de una distribución del conjunto entrenamiento es la siguiente

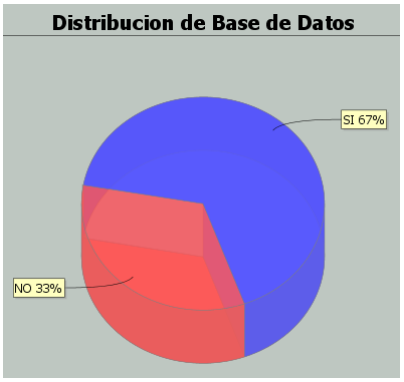
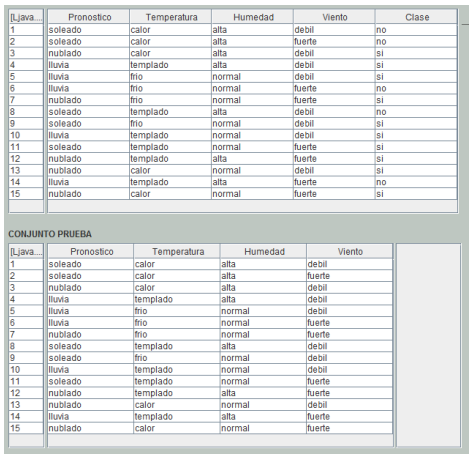



Figura 3.4.1 Ejemplo de Distribucion de clases

III. **Visualización del conjunto entrenamiento y conjunto prueba:** Se creó una sección para la visualización del conjunto prueba y conjunto entrenamiento por medio de tablas (Véase la figura 3.5).


**Figura 3.5 Sección de tablas para Entrenamiento y Prueba**

En esta seccion se vizualizaran los archivos cargados, en la parte superior se visulizara el conjunto entrenamiento y en la parte inferior el conjunto prueba. (Figura 3.6)



| [Ljava... | Pronostico | Temperatura | Humedad | Viento | Clase |
|---|---|---|---|---|---|
| 1 | soleado | calor | alta | debil | no |
| 2 | soleado | calor | alta | fuerte | no |
| 3 | nublado | calor | alta | debil | si |
| 4 | lluvia | templado | alta | debil | si |
| 5 | lluvia | frio | normal | debil | si |
| 6 | lluvia | frio | normal | fuerte | no |
| 7 | nublado | frio | normal | fuerte | si |
| 8 | soleado | templado | alta | debil | no |
| 9 | soleado | frio | normal | debil | si |
| 10 | lluvia | templado | normal | debil | si |
| 11 | soleado | templado | normal | fuerte | si |
| 12 | nublado | templado | alta | fuerte | si |
| 13 | nublado | calor | normal | debil | si |
| 14 | lluvia | templado | alta | fuerte | no |
| 15 | nublado | calor | normal | fuerte | si |

CONJUNTO PRUEBA

| [Ljava... | Pronostico | Temperatura | Humedad | Viento |
|---|---|---|---|---|
| 1 | soleado | calor | alta | debil |
| 2 | soleado | calor | alta | fuerte |
| 3 | nublado | calor | alta | debil |
| 4 | lluvia | templado | alta | debil |
| 5 | lluvia | frio | normal | debil |
| 6 | lluvia | frio | normal | fuerte |
| 7 | nublado | frio | normal | fuerte |
| 8 | soleado | templado | alta | debil |
| 9 | soleado | frio | normal | debil |
| 10 | lluvia | templado | normal | debil |
| 11 | soleado | templado | normal | fuerte |
| 12 | nublado | templado | alta | fuerte |
| 13 | nublado | calor | normal | debil |
| 14 | lluvia | templado | alta | fuerte |
| 15 | nublado | calor | normal | fuerte |

**Figura 3.6 Entrenamiento y prueba cargados**

Nótese que la última columna está vacía, esto es porque la columna está reservada para la clasificación, en esta columna se propondrá una clase para cada una de las instancias prueba, se llenara cuando se ejecute la clasificación.

**IV. Información de la clasificación:** En esta sección de la ventana principal mostrara la medida de funcionalidad y el tiempo que llevo al algoritmo la clasificación. Mostrar esta información nos ayudara a sacar conclusiones y determinar que clasificador es más adecuado para el repositorio del Reactor de Microactividad de Cracking Catalítico.

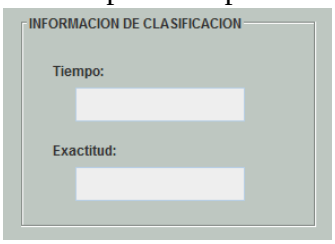



Figura 3.7 Información de clasificación

**V. Opciones de test:** En esta sección se mostraran las opciones con las que se pretende poner a prueba a los clasificadores, se proponen 4 diferentes opciones las cuales son:

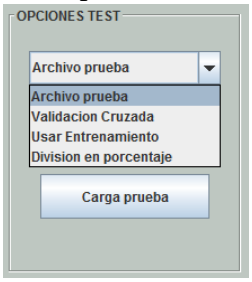
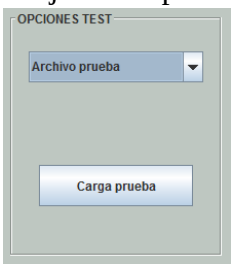
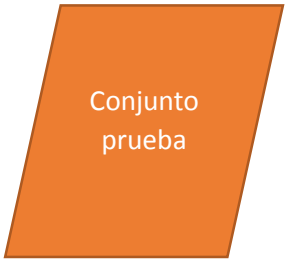



Figura 3.8 Opciones de Test

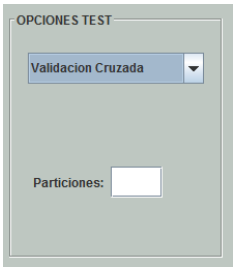
- **Archivo Prueba:** se da la opción de cargar un archivo externo para representar el conjunto de prueba.

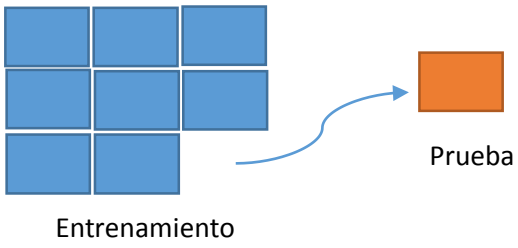




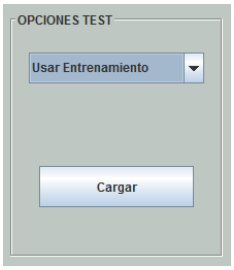



- **Validación Cruzada:** esta técnica permite realizar n particiones al conjunto entrenamiento, agarrar n-1 particiones para el conjunto entrenamiento y la partición restante como conjunto prueba, pero esto se debe hacer n veces tal que el conjunto prueba haya agarrado todas las particiones una vez. Esta prueba puede garantizar que son independientes de la partición entre datos de entrenamiento y prueba.

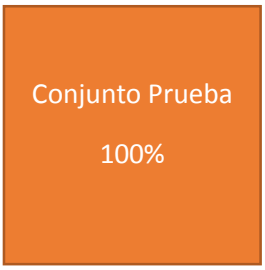


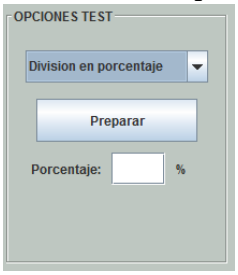

- **Usar Entrenamiento:** se da la opción de usar el 100 % del conjunto de entrenamiento como conjunto prueba.

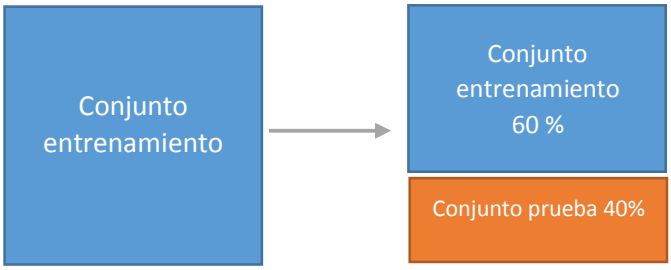







- **División por porcentaje:** Se da la opción de usar un cierto porcentaje del conjunto entrenamiento para el conjunto prueba, la selección de los ejemplo en el conjunto entrenamiento para el conjunto prueba es aleatoria.



VI. **Clasificadores:** En esta sección de la ventana principal se muestra una lista desplegable con los 3 diferentes clasificadores vistos en la sección 2 los cuales son; Bayes Ingenuo, KNN e ID3, estos clasificadores necesitan el conjunto entrenamiento y el conjunto prueba ya cargados en la aplicación.

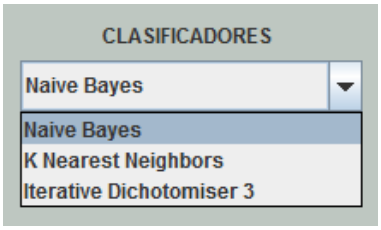



**Figura 3.9 Lista de Clasificadores**

VII. **Permutar celdas:** Este botón en la parte superior es el que se encarga de permutar las celdas de los ejemplos de entrenamiento que pertenecen a una determinada clase, esto con el objetivo de crear una base de datos más grande y generar conocimiento implícito.


Figura 3.10 Permutar celdas

VIII. **Clasificar:** Este botón es el que se encarga de empezar la clasificación con el clasificador seleccionado en la lista desplegable.


Figura 3.11 Botón Clasifica

**Arquitectura de la implementación**

En esta sección se expondrá como se trabajó la arquitectura de la aplicación, cuáles fueron los modelos utilizados y los flujos principales.

El patrón de arquitectura de software, que se utilizo fue; El modelo-Vista-Controlador.

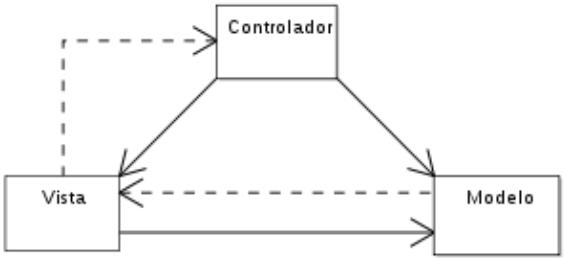
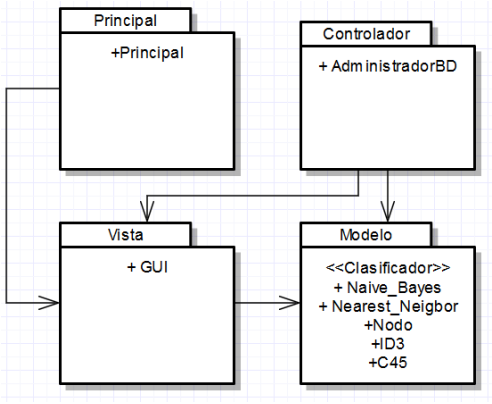



Figura 3.12 Modelos Vista Controlador

La arquitectura MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario, la principal característica de este modelo es que se separa la lógica de negocio con el diseño esto permite al proyecto ser escalable.

Para visualizar la estructura del programa y observar cómo se distribuyeron las clases con el modelo MVC se presentara el Diagrama de paquetes.

Figura 3.13 Diagrama de paquetes




Un diagrama de paquetes en el lenguaje unificado de modelado representa las dependencias entre los paquetes que componen un modelo, es decir muestra como un sistema está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones.

El programa contiene 4 paquetes;

1. **Principal:** El paquete principal contiene el hilo principal de la ejecución, la clase principal iniciara la ejecución de la interfaz gráfica de usuario.
2. **Controlador:** El paquete controlador contendrá el administrador de base de datos, el administrador es el que se encarga de suministrar métodos necesarios para la manipulación y gestionar las bases de datos.

3. **Vista:** El paquete vista se encargara de lanzar la interfaz gráfica, la interfaz ayudara a visualizar los datos, la distribución del conjunto de entrenamiento, opciones de clasificación y resultados de la clasificación.
4. **Modelo:** El paquete modelo, es el principal objetivo de la implementación, en este paquete se encuentran los algoritmos de clasificación, estos son los que se encargan de clasificar las bases de datos para agrupar los datos a una clase propuesta.

Diseñar la estructura de la aplicación es importante, pero también debemos de revisar cual es el comportamiento de la aplicación, es decir, cuales son los flujos que la aplicación sigue para su funcionamiento, para ello se realizó una diagrama de secuencia. (Véase Figura 3.14)

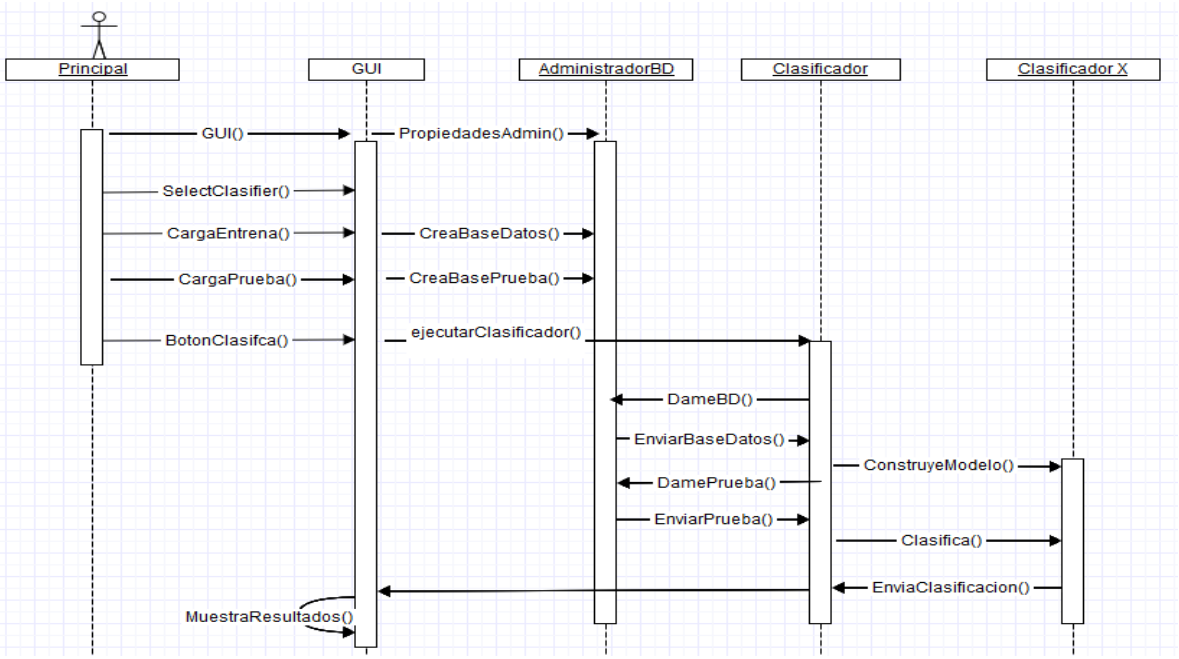



Figura 3.14 Diagrama de secuencia

Un diagrama de secuencia es un tipo de diagrama usado para modelar interacción entre objetos en un sistema, esto nos ayuda a seguir el comportamiento de la aplicación y así comprenderlo fácilmente.

A continuación se detallaran los elementos más importantes del diagrama de secuencia:

**Interaccion con el usuario:** El usuario es el que se encarga iniciar la aplicación, como tambien la creacion del Administrador de Base de Datos.(Vease la figura 3.15)

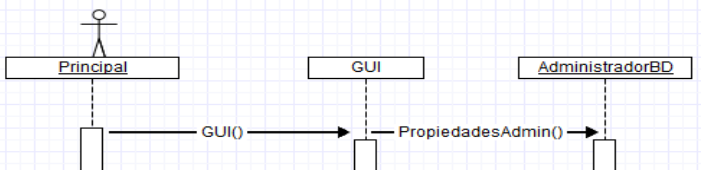



**Figura 3.15 Iniciar programa**

**Selección de un clasificador**: El flujo normal sigue su curso y debemos de seccionar un algoritmo de clasificación. (Véase Figura3.16)

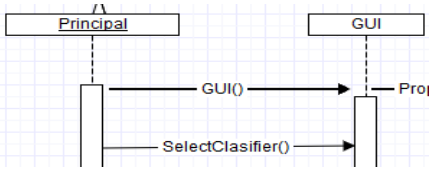



**Figura 3.16 Seleccionar clasificador**

**Cargar Base de datos:** se cargan las bases de datos, al momento de darle click al boton "carga entrenamiento" aparecera un buscador de archivo en donde se tendra que seleccionar el archivo que representa el conjunto entrenamiento, despues de seleccionar el entrenamiento el administrador gestiona el archivo para convertirlo informacion utilizable para la aplicación. Despues de tener el conjuto entrenamiento se tendra que seleccionar el conjunto prueba con alguna de las opciones de "test de prueba" (Vease Figura 3.8)

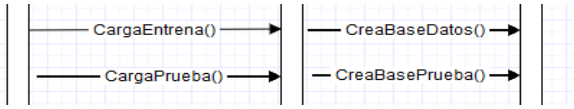



**Figura 3.17 Cargar bases de datos**

**Clasificacion:** Ya con el conjunto entrenamiento y el conjunto prueba cargados en el sistemas, se prosigue a clasificar las instancias prueba, esto se hace seleccionando algun clasificador de la lista desplegable (Vease figura 3.9), cuando se selecciónó el clasificador K vecinos mas cercanos aparecera un campo de texto en el cual tenemos que introducir una constante k (Figura 3.18) que representara el numero de k vecinos que tenemos que considerar para determinar a que clase pertenece la instancia prueba por medio de la votacion, sin embargo para las otras opciones de clasificadores no se re quiere nada mas.

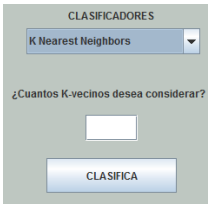



**Figura 3.18 Seleccionar KNN**

Ya selecionado el clasificador le damos click en el boton clasificar (Figura 3.11), al momento de darle en clasificar el programa realizara las preparaciones iniciales para la clasificacion despues el administrador de Bases de datos ejecuta el algoritmo del clasificador seleccionado.

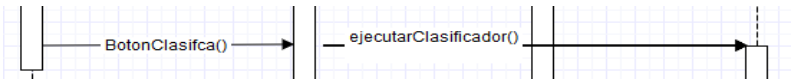



**Figura 3.19 Ejecución del clasificador**

**Fase de entrenamiento:** En esta parte del diagrama de secuencia se presenta la forma en la que realiza la fase de entrenamiento, aquí es en donde se crea el modelo de clasificacion.K vecinos mas ceranos es una excepcion a esto,ya que como tal no se alcanza a distinguir una linea divisora entre la fase de entrenamiento con la fase de prueba.

Primero el clasificador pide el conjunto entrenamiento, el administrador de base de datos le responde enviando la base de datos, el clasificador construye el modelo de clasificacion el cual nos servara para proponer una clase a las instancias prueba. (Vease figura 3.20)

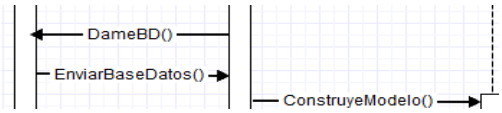



**Figura 3.20 Fase de entrenamiento**

**Fase de clasificacion:** En esta fase se pretende etiquetar a las instancias prueba, para ello el clasificador ya tiene el modelo de clasificacion, el siguietne paso es pedir las instancais prueba al adminsitrador de base de datos, el administrador le responde enviandole el conjunto de prueba, una vez que el clasificador tenga el conjunto entrenamiento y el conjunto prueba se continuara con el proceso de clasificacion, para ello el clasificador dispone de diferentes algoritmos dependiendo del clasificador seleccionado.

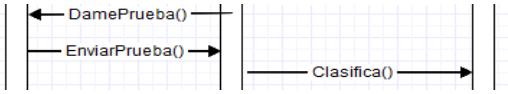



**Figura 3.21 Fase de clasificación.**

**Enviar resultados:** Ya hecha la clasificación se prosigue a enviar las clases propuestas a la GUI para que muestre los resultado, estos resultados se dividen en 3 tipos; Medidas de funcionalidad, tiempo y columna de clasificación.

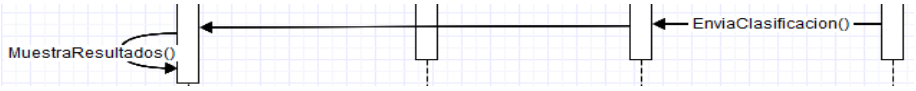



**Figura 3.22 Mostrar resultados**

De forma muy general así es como funciona el programa, sin embargo hay detalles que pasamos por alto, pueden surgir algunas preguntas como por ejemplo, ¿cómo funciona la clase clasificador?, ¿Cómo es la relación de la clase clasificador con los 3 tipos de clasificadores?, ¿Cómo funcionan los 3 diferentes tipos de clasificadores? Los detalles de estas preguntas se detallaran en el siguiente apartado.

**Estructura de la clase clasificador**

Como observamos en el diagrama de secuencia hay algunas clases que tiene tareas importantes y fundamentales en la aplicación, una de ellas es la clase clasificador, como se

explicó en el apartado anterior la clase clasificador es la que se encarga de realizar la clasificación en base a la información recibida, El diagrama de clases de esta clase es el siguiente;

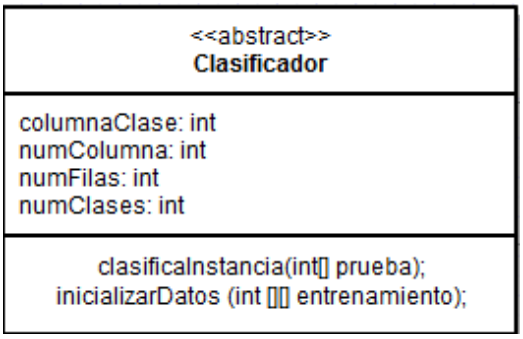



**Figura 3.23 Figura clase Clasificador**

Como vemos la clase clasificador es una clase abstracta, recordemos que una clase abstracta en una clase en la que alguno de sus métodos está declarado pero no está definido, es decir, se especifica su nombre, parámetros y tipo de devolución pero no incluye código.

Una clase abstracta nos sirve para satisfacer el objetivo del clasificador, ya que puede comportarse de acuerdo al clasificador elegido, esto da cierta flexibilidad a la hora de ahorrar líneas de código en cada uno de los clasificadores.

Los dos métodos que se observan en el diagrama de clases son abstractos, estos se pueden comportar de diferente manera de acuerdo del clasificador elegido.

La figura 3.24 muestra como es la relación entre los clasificadores y la clase clasificador.

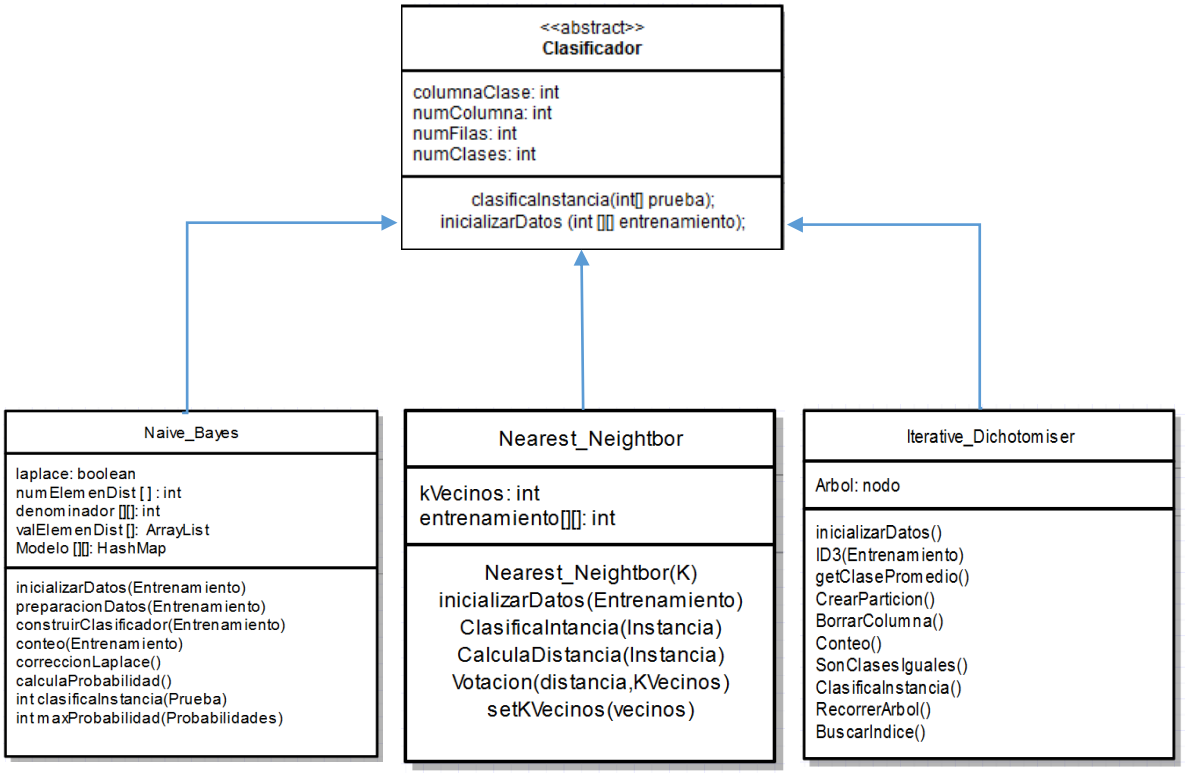



**Figura 3.24 Estructura de la clase clasificador**

En la figura 3.24 podemos ver que se utiliza uno de los conceptos más importantes y valiosos de la programación orientada a objetos, la herencia, podemos definir la herencia como la capacidad de crear clases que adquieran de manera automática los miembros de otras clases que ya existen, pudiendo al mismo tiempo añadir atributos y métodos propios.

Esta definición se aplicó en la estructura de la clase clasificador, esto es sumamente importante ya que esto permitió que las subclases Naive_Bayes, Nearest_Neightbor e Iterative_Dichotomiser heredaran los métodos y atributos de la clase clasificador, la ventaja principal de usar herencia es la reutilización de código, es importante en aquellos casos donde se necesita crear métodos que se comportan de igual manera.

A continuación se detallaran implementaciones de tres clases, estas tres clases son las más importantes de la implementación;

- Bayes Ingenuo
- K vecinos más cercanos
- ID3

# 3.3. Implementación de los Clasificadores

La clase Clasificador es una clase abstracta, como dijimos anteriormente una clase abstracta es una clase en la que alguno de sus métodos es abstracto. Un método se define como abstracto porque en ese momento no se conoce como ha de ser su implementación, serán las subclases de la clase abstracta las responsables de darle cuerpo mediante la sobrescrita del mismo.

La idea principal de poner la clase clasificador como clases abstracta es usar polimorfismo, El polimorfismo se basa en gran medida en conceptos como herencia, sobrescritura de métodos y supone el principal motivo de la existencia de las clases abstractas.

En java, es posible asignar un objeto de una clase a una variable de su superclase. Esto es aplicable, incluso, cuando la superclase es una clase abstracta.

Entrando en contexto por ejemplo, dada una variable de tipo Clasificador:

Clasificador clasifi;

Es posible asignar a esta variable un objeto Naive_Bayes:

clasifi = new Naive_Bayes (…);

A partir de aquí, **puede utilizarse esta variable para invocar a aquellos métodos del objeto que también estén definidos o declarados en la superclase,** pero no a aquellos que solo existan en la clase a la que pertenece el objeto.

Por ejemplo (Vease la figura 3.24), puede utilizarse la variable clasifi para invocar a los métodos ClasificaInstancia(…); o InicializarDatos(…); del objeto Clasificador, pero no para llamar a conteo() o calculaProbabilidad() del objeto Naive_Bayes.

Posiblemente, nos estemos preguntando ¿Qué utlidad puede tener asignar un objeto a una variable de su superclase para llamar a sus métodos, cuando eso mismo podemos hacerlo si le asignamos a una variable de su propia clase?

Para responder a esta pregunta, volvamos al ejemplo anterior, además imaginemos que tenemos también la clase Nearest_Neightbor y iterative_Dichotomiser como subclases de Clasificador. Según lo comentado en el apartado anterior, seria posible almacenar en una variable Clasificador cualquier objeto de sus subclases, es decir objetos Naive_Bayes , Nearest_Neightbor y iterative_Dichotomiser:

```
//variable de la superclase
        Clasificador clasifi;
        Clasifi = new Naive_Bayes(…);
        Clasifi.ClasificaInstancia(…); //Metodo ClasificaInstancia de Naive_Bayes
        Clasifi = new Naive_Bayes(…);
        Clasifi.ClasificaInstancia(…); // Metodo ClasificaInstancia de Nearest_Neightbor
        Clasifi = new iterative_Dichotomiser();
```

Clasifi.ClasificaInstancia(…); // Metodo ClasificaInstancia de iterative_Dichotomiser

De lo anterior se desprende un hecho muy interesante: la misma instrucción Clasifi.ClasificaInstancia(…); permite llamar a distintos métodos ClasificaInstancia(…), dependiendo del objeto almacenado en la variable Clasifi.

En esto consiste precisamente la idea principal que puede definirse de la siguiente manera, "La posibilidad de utilizar una misma expresión para invocar a diferentes versiones de un mismo método, determinando en tiempo de ejecución la versión del método que se debe ejecutar".

El siguiente código muestra cómo y dónde se utilizó la idea anterior

```
private void accionClasificar() {
        switch (comboClasificador.getSelectedIndex()){
        case 0:
                //Clasificador bayes ingenuo.
                Clasificador bayesIngenuo = new Naive_Bayes();
                ejecutarClasificador(bayesIngenuo);
                break;
        case 1:
                //Clasificador K-Vecinos.
                //Gestionamos el Valor K (Omisíon del bloque de código).
                Clasificador kVecinosCercanos = new Nearest_Neightbor(k);
                ejecutarClasificador(kVecinosCercanos);
                break;
        case 2:
                Clasificador iterativeD3 = new Iterative_Dichotomiser();
                ejecutarClasificador(iterativeD3);
                break;
        }

}
```

Nos podemos dar cuenta en las líneas subrayadas con amarillo que se crean diferentes objetos de clasificadores pero se guardan en una variable de tipo Clasificador, después se llama un solo método que puede ser utilizado por los tres objetos, eso es una aplicación de lo que se le conoce como Polimorfismo.

## 3.3.2. Implementación de Bayes Ingenuo

En este apartado se explicara cómo se implementó el clasificador bayes ingenuo y cuáles fueron las estrategias que se siguieron para llevarlo a cabo.

Como se vio en la teoría, los clasificadores se dividen en dos fases; la etapa de construcción y la etapa de clasificación, a continuación veremos cómo se implementó la etapa de construcción.

**Etapa de construcción**

En la figura 3.25 podemos observar una serie de rectángulos metidos unos con otros, estos triángulos representan métodos, donde si un rectángulo está dentro de otro significa que un método invoca otro método.

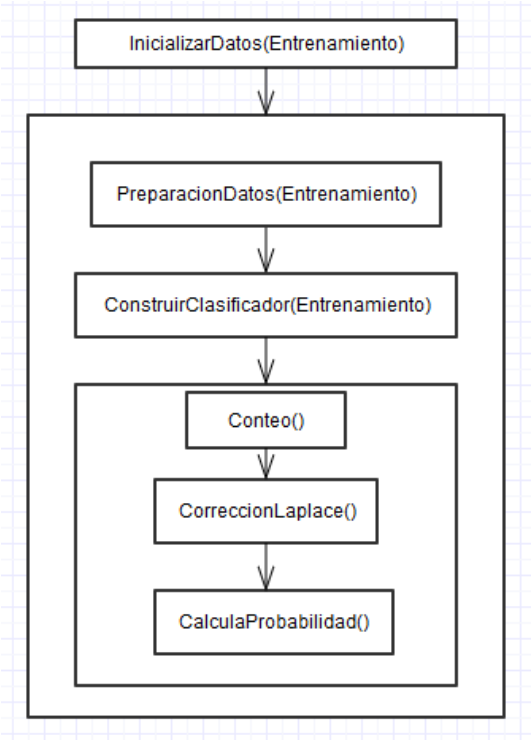



**Figura 3.25 Algoritmo de Bayes Ingenuo**

El método principal es **InicializarDatos(Entrenamiento)**, el objetivo de este método es preparar lo necesario para que se realice la clasificación, esto implica crear el modelo probabilístico donde se guardaran todas las probabilidades a priori y condicionales.

**Nivel Código:**

```
public void inicializarDatos(int [][] entrenamiento){
        //Comenzamos sacando los datos necesarios
        preparacionDatos(entrenamiento);
        //Crear el modelo Probabilistico.
        construirClasificador(entrenamiento);
```

```
}
```

El método **PreparacionDatos(Entrenamiento)** se encarga se general los parámetros iniciales como lo son;

- Numero de filas.
- Numero de columnas.
- Numero de clases.
- Valores distintos de un atributo.

**Nivel Código:**

```
private void preparacionDatos(int [][] entrenamiento){
        //Declaracion e Inicializacion de datos.
        //Dimensiones
        numFilas= entrenamiento.length;
        numColumnas= entrenamiento[0].length;
        //Variable que indica el indice del atributo clase.
        columnaClase= numColumnas-1;
        //Arreglo que guardara el numero de elementos distintos de cada columna.
        numElemDist = new int [numColumnas];
        //Arreglo de listas que guardara todos los valores distintos de cada columna.
        valElemDist = new ArrayList[numColumnas];
        //Obtenemos los elementos diferentes de cada atributo.
        for (int columna = 0; columna < numColumnas; columna++){
                valElemDist[columna] = dameValorDist(entrenamiento,columna);
                numElemDist[columna] = valElemDist[columna].size();
        }
        //Obtenemos el numero de clases
        numClases= numElemDist[columnaClase];
        //HashMap que representara el modelo probabilistico.
        modelo = new HashMap [numClases][numColumnas];
        //Denominador para sacar las probabilidades de la formula de Naive bayes.
        denominador= new int [numClases][numColumnas];
}
```

El código muestra cómo se van generando los datos iniciales. Una variable Importante es valElemDist = **new** ArrayList[numColumnas];

En ella se van a guardar todos los valores que puede tomar los atributos, dependiendo de qué atributo sea se guardaran en un arrayList. El método que hace la acción de sacar los valores de una columna es *dameValorDist*(entrenamiento,columna); un método muy importante no solo en bayes ingenuo sino en ID3.

La estructura más importante es modelo = **new** HashMap [numClases][numColumnas]; porque es en esta variable donde se guardaran las probabilidades a priori y condicionales, es importante recalcar que la estructura en donde guardaran en una matriz de HasMap, un Hasmap es un objeto en el que puedes guardar cualquier tipo de objeto, asignándole una key,

para más tarde, poder recuperar ese objeto mediante la key. Nos sirve mucho, ya que la idea es que solamente le demos el atributo y nos arroja la probabilidad.

Con estos datos iniciales se comienza a trabajar con el siguiente método.

El método **construirClasificador()** es el encargado de la construcción del modelo probabilístico, pero para lograr eso tendrá que realizar algunas subtareas indispensables;

- Conteo()
- CorrecciónLaplace().
- Calcular Probabilidades().

**Nivel Código**

```
public void construirClasificador(int [][] entrenamiento){
        //Calculamos el conteo de elementos.
        conteo(entrenamiento);
        //Realizamos la correccion de laplace, si es necesario.
        if(laplace) correccionLaplace();
        //Calculamos las probabilidades.
        calculaProbabilidad();
}
```

Primero realizamos el conteo para las probabilidades a priori y condicionales en la estructura Modelo que creamos anteriormente.

La instrucción **if**(laplace) correccionLaplace(); es muy fácil de intuir, en dado caso que en la tabla haya un cero entonces ejecuta la corrección de Laplace, esto se mantiene controlado por una bandera, una variable boolean llamada Laplace.

La ultima instrucción se calcula la probabilidad a priori y condicional y se guarda en el modelo.

El método **conteo()** tiene la tarea de realizar el conteo para los cálculos de las probabilidades a priori y condicionales, como se vio en la sección teórica 2.(Véase tabla 2.15)

**Nivel Código**

```
private void conteo(int [][] entrenamiento){
        //Comenzamos con el conteo de elementos
        for (int clase = 0; clase < numClases ; clase++){
                for (int columna = 0; columna < numColumnas; columna++) {
                        //Creamos un HashMap para cada atributo de cada clase.
                        modelo[clase][columna] = new HashMap <Integer, Double>();
                        //Permitira acumular el conteo de valores de cada atributo de
                        cada clase.
                        int suma= 0;
                        for (int k = 0; k < numElemDist[columna]; k++) {
                                //Sacaremos el conteo de cada valor distinto de cada
                                atributo de cada clase.
                                //Tomamos un valor distinto del atributo.
```

```
                                    int valor = valElemDist[columna].get(k);
                                    //Tomamos el valor de clase.
                                    int                                 valorClase=
valElemDist[columnaClase].get(clase);
                                    //Hacemos el conteo
                                    double                              conteo=
                                    contar(entrenamiento,columna,valor,valorClase);
                                    //Agregamos el valor al HashMap con su respectivo
                                    conteo.
                                    modelo[clase][columna].put(valor, conteo);
                                    //Suma parcial de conteo que cumple con el valor y
                                    clase especificada.
                                    suma += conteo;
                            }
                            //Distincion entre el conteo a priori y el conteo parcial de los
                            atributo clase.
                            if (columna == columnaClase)
                                    denominador[clase][columna] = numFilas-1;
                            else denominador[clase][columna] = suma;
                    }
            }
}
```

En términos muy generales, se crearon 3 for ya que dos recorrerán el conjunto entrenamiento y uno con ayuda de los 2 for anteriores recorrerán el modelo Hasmap y realizaran el conteo con un método contar(entrenamiento,columna,valor,valorClase); un bloque de instrucción importante a considerar son las últimas instrucciones donde se calculan los denominadores de las probabilidades.

El método **CorrecciónLaplace()**, es un método que corrige una falla del algoritmo de bayes ingenuo, ejemplo

$$P(si|soleado, fria, alta, cierto) = p(si)p(soleado|si)p(fria|si)p(alta|si)p(cierto|si)$$

Aquí podemos observar que para calcular la probabilidad a posteriori cuando estamos realizando una multiplicación de probabilidades condicionales y a priori. Sin embargo hay un error que nos puede causar problemas, el cual es cuando algunas de las probabilidades que se están multiplicando sea cero, el principal problema de que la probabilidad se haga cero es que se pierde información ya que con una que haga cero todas las demás probabilidades se vuelven cero, entonces se pretende corregir este problema. Cuando el problema surge es común que sea por las siguientes causas;

- **En la estructura de tabla de conteo se tiene un cero:** esto hace que la probabilidad por ejemplo $p(soleado|si) = \frac{0}{14} = 0$ , entonces al multiplicar por las otras probabilidades se pierda toda la información acumulada, la forma de resolver esto es por medio de un método llamado la corrección suave de Laplace, donde simplemente se le suma 1 a todo conteo, así evitando que haya ceros.

Si vemos la tabla 2. podemos observar que hay un cero en el conteo en el atributo Cubierto de la clase No, en este caso causaría problemas si nos pidieran calcular la probabilidad condicional $p(cubierto|no) = \frac{0}{5} = 0$, por tanto sumamos 1 a toda la tabla de conteo quedando como la tabla 2.1, $p(cubierto|no) = \frac{1}{4+1+3} = \frac{1}{8} = 0.125$

Observe que no se le sumo al conteo de las probabilidades a priori (La última fila), no es necesario porque estas nunca serán cero.

| Clase SI | | Clase NO | |
|---|---|---|---|
| Pronostico | | Pronostico | |
| Soleado | 3 | Soleado | 4 |
| Cubierto | 5 | Cubierto | 1 |
| Lluvioso | 4 | Lluvioso | 3 |
| Temperatura | | Temperatura | |
| Alta | 3 | Alta | 3 |
| Suave | 5 | Suave | 3 |
| Fría | 4 | Fría | 2 |
| Humedad | | Humedad | |
| Alta | 4 | Alta | 5 |
| Normal | 7 | Normal | 2 |
| Viento | | Viento | |
| Cierto | 4 | Cierto | 4 |
| Falso | 7 | Falso | 3 |
| Lluvia | | Lluvia | |
| Si | 9 | No | 5 |

Tabla 2.1 Corrección de Laplace

**Nivel Código**

```
private void correccionLaplace(){
        //Incrementamos uno a todos los valores del HashMap, es la corrección suave
        de laplace.
        for (int clase = 0; clase < numClases ; clase++){
                //No se incrementan las Probabilidades a priori.
                for (int columna = 0; columna < columnaClase; columna++){
                        int suma= 0;
                        for (int k = 0; k < numElemDist[columna]; k++) {
                                int llave= valElemDist[columna].get(k);
                                double                                  valorActual=
modelo[clase][columna].get(llave);
                                //Sumamos y actualizamos los valores del HashMap
                                modelo[clase][columna].replace(llave,
++valorActual);
                                suma += valorActual;
                        }
```

```
                    //Actualizamos el denominador
                    denominador[clase][columna] = suma;
                }
            }
        }
```

Se puede apreciar cómo se recorre todo otravez y en la línea modelo[clase][columna].replace(llave, ++valorActual);

Se incrementa en uno el valor actual. Hay una variable suma que acumula los nuevos valores para actualizar el denominador.

Después de la corrección de Laplace, ahora si podemos calcular las probabilidades, el método destinado para esta tarea es **CalculaProbabilidad(),** este método calcula las probabilidades condicionales y a priori, y las guarda en el modelo probabilístico.

Tenemos un problema al calcular las probabilidades, porque **Las probabilidades pueden hacerse muy pequeñas hasta interpretarse como cero en la máquina,** ya que las probabilidades son cantidades que están en el intervalo de [0,1] por lo tanto en la mayoría de las veces las probabilidades sean números menores a 1 esto provoca que cada vez que se multiplique un numero fraccionario se haga más chico la probabilidad haciendo que la computadora después de un cierto número de cifras significativas la maquina no puede soportar y las interpreta como cero, teniendo el problema de perder información. Una solución a este problema es la utilización de logaritmos, con ayuda de las propiedades de los logaritmos logramos transformar los números pequeños de las probabilidades a número más grandes. Las propiedades de logaritmos utilizadas son las siguientes:

$$\log\frac{A}{B} = \log A - \log B$$

$$\log AB = \log A + \log B$$

Así logramos convertir las probabilidades muy pequeñas a números grandes

$$\frac{A}{B} * \frac{C}{D} \quad \rightarrow \quad \log A - \log B + \log C - \log D$$

**Ejemplo**

$$\frac{2}{4} * \frac{3}{5} = 0.3 \quad \rightarrow \quad \log 2 - \log 4 + \log 3 - \log 5 = -0.5228$$

Se puede apreciar un poco la diferencia

**Nivel Código**

```
public void calculaProbabilidad(){
        for (int clase = 0; clase < numClases ; clase++){
                for (int columna = 0; columna < numColumnas; columna++) {
                        for (int k = 0; k < numElemDist[columna]; k++){
                           int llave= valElemDist[columna].get(k);
                           double valorActual= modelo[clase][columna].get(llave);
```

```
                        //Aplicacion de Logaritmo Natural, en lugar de la division.
                            double probabilidad= Math.log(valorActual)-
                            Math.log(denominador[clase][columna]);
                          modelo[clase][columna].replace(llave, probabilidad);
                        }
                    }
                }
            }
```

Los 3 for en el código anterior sirven para recorrer todo el modelo y sacar el conteo correspondiente, podemos notar como la instrucción

```
double probabilidad= Math.log(valorActual)- Math.log(denominador[clase][columna]);
```

Hace uso de los logaritmos para prevenir la perdida de información.

**Etapa de clasificación**

Anteriormente en la etapa de construcción se explicó cómo se interpretaba el diagrama de la Figura 3.26, la etapa de clasificación es corta en comparación con la etapa de entrenamiento, en esta etapa se tiene la ventaja de que las probabilidades ya están calculadas y solo basta con buscarlas en el modelo.

El método **ClasificaInstancia(instancia)** tiene la tarea de calcular las probabilidades a posteriori, ya que se tienen que comparar al final. Se tienen tantas probabilidades a posteriori como numero de clases, el parámetro de entrada representa las instancias prueba que se tienen que clasificar, este método crea un arreglo con las probabilidades a comparar.

**Nivel Código**

```
public int clasificaInstancia(int[] prueba){
    int numAtributos= prueba.length;
    //Arreglo; Guarda las probabilidades de cada clase.
    double probabilidades []= new double [numClases];
    //Suministramos las probabilidades del modelo probabilistico.
    for(int clase= 0; clase< numClases; clase++){
        //comenzamos con 1 para no tener problemas en la multiplicacion.
        probabilidades[clase]= 0.0;
        for(int columna= 0; columna<numAtributos; columna++){
            //sacamos los valores de la instancia
            int llave= prueba[columna];
            //Se controla cuando el valor de la instancia no se encuentra en
            el modelo.
            if(modelo[clase][columna].containsKey(llave)){
            //Con ayuda del Logaritmo Natural sumamos las
            probabilidades condicionales P(a|C) , P(b|C) ...
```

```
                        probabilidades[clase]                    +=
                        modelo[clase][columna].get(llave);
                    }else{
                        //Si no se encuentra el valor en el modelo, se calcula su
                        probabilidad; 1/(total+1)
                        //Ln(1)-Ln(total+1) = 0-Ln(total+1) = -Ln(total+1)
                        probabilidades[clase]                    -=
                        Math.log((denominador[clase][columna]+1));
                    }
                }
                //Con ayuda del Logaritmo Natural sumamos las probabilidades a
                priori de la clase P(C)
                int valor = valElemDist[columnaClase].get(clase);
                probabilidades[clase] += modelo[clase][columnaClase].get(valor);
            }

            //Pedimos la clase con mayor probabilidad.
            int indiceMAX= maxProbabilidad(probabilidades);

            //con el indice de la Clase mas Probable sacamos la clase.
            int ganadora = valElemDist[columnaClase].get(indiceMAX);

            return ganadora;
        }
```

El código anterior puede resultar bastante grande pero es similar a todos los demás ya que solo se recorre el modelo probabilístico saca el valor y el denominador, y se realiza la probabilidad por medio de logaritmos, las probabilidades resultantes se guardan en un arreglo: probabilidades[clase]
El método **MaxProbabilidad()** busca la mayor de las probabilidades a posteriori de su clase correspondiente, esto con el fin de elegir la clase más probable y poder proponer una clase para la instancia prueba correspondiente.

Lo que prosigue es sacar el índice de la probabilidad más grande
`int indiceMAX= maxProbabilidad(probabilidades);`

Si sabemos el índice entonces sabemos la clase. La última instrucción retorna la clase que bayes ingenuo propone a la instancia prueba.

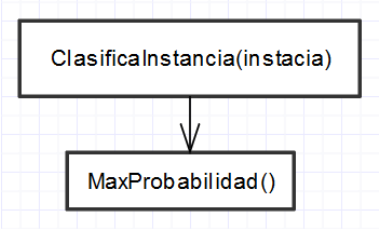


**Figura 3.26 Etapa de clasificación**

Está claro que bayes ingenuo resulta un clasificador no difícil de programar, y cuenta con algunos detalles que se tienen que tratar con cuidado, pero no pasa de solo tener que contar y calcular probabilidades, después Tendremos la oportunidad de comparar los demás clasificadores ya implementados con bayes ingenuo.

A continuación se pondrá el diagrama de clases, para saber el contenido y el comportamiento de la clase, en ella contienen todos los métodos y atributos, nos permitirá saber es el estado y el comportamiento.

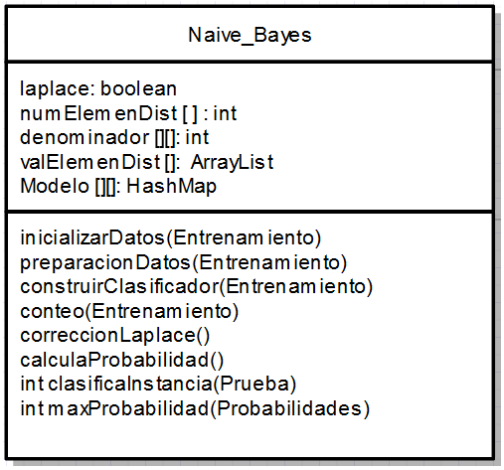



**Figura 3.26.1 Diagrama de clase de Bayes Ingenuo**

## 3.3.3. Implementación de K Nearest Neighbor

En esta sección se detallara el algoritmo k vecinos más cercanos, los métodos de programación y estructuras de implementación que se obtuvieron para la realización de este clasificador. La teoría nos indica que K vecinos más cercanos es un algoritmo de clasificación que se sigue utilizando en algunas áreas de investigación, tiene ramas de investigación muy interesantes como por ejemplo una forma o técnica de reducir los costos en cómputo y tiempo. En este apartado vamos a descubrir si K Vecinos más cercanos es un clasificador difícil de implementar.

Una característica que distingue a K vecinos más cercanos de otros algoritmos de clasificación es el número de etapas que se consideran para la implementación de dicho algoritmo, en KNN no hay una distinción clara de la división entre la fase prueba y la fase entrenamiento, entonces se va a considerar en una sola etapa la implementación de este algoritmo.

En la figura 3.27 mostramos el algoritmo KNN, donde los rectángulos representan métodos, se explicaran las partes de este algoritmo.

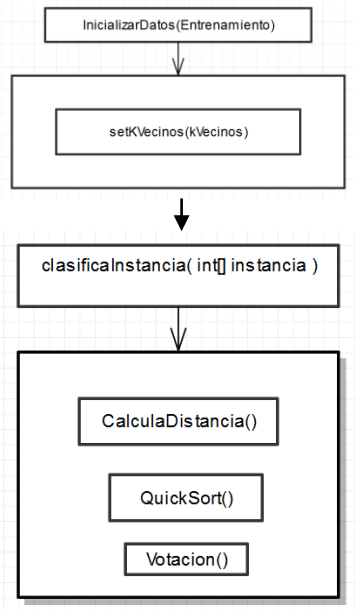



**Figura 3.27 Algoritmo K NN**

El método inicializarDatos(Entrenamiento), es un método encargado de definir y crear las variable iniciales del algoritmo, algunas de las variables indispensables que se inicializan en este método son;

- Numero de filas.
- Numero de columnas.
- Numero clases.
- K vecinos.

**Nivel Código**

```
public void setKVecinos(int vecinos){
        if(vecinos>0 && vecinos < numFilas-1)
                kVecinos = vecinos;
        else{
                JOptionPane.showMessageDialog(null,
                "El valor de K-Vecinos no esta en intervalo adecuado."
                + "\nTiene que ser: \n"
                +"- mayor a 0\n- menor a "+(numFilas-1)+
                ".\nNOTA: Se pondra el valor por default: K= 1",
                "ERROR",JOptionPane.ERROR_MESSAGE);
                kVecinos=1;
        }
}
```

El método setKVecinos(KVecinos) es el encargado de inicializar la constante k, la constante k es el número de vecinos que son considerados para la votación final para determinar si una instancia prueba pertenece a una clase propuesta, la inicialización de este atributo es importante, permite controlar errores de inicialización e impide crear inconsistencias. El rango de este atributo está controlado por el número filas. En dado caso que se introduzca un k no valido, es mandado un mensaje avisando que el valor de k no es correcto, Véase la siguiente imagen.

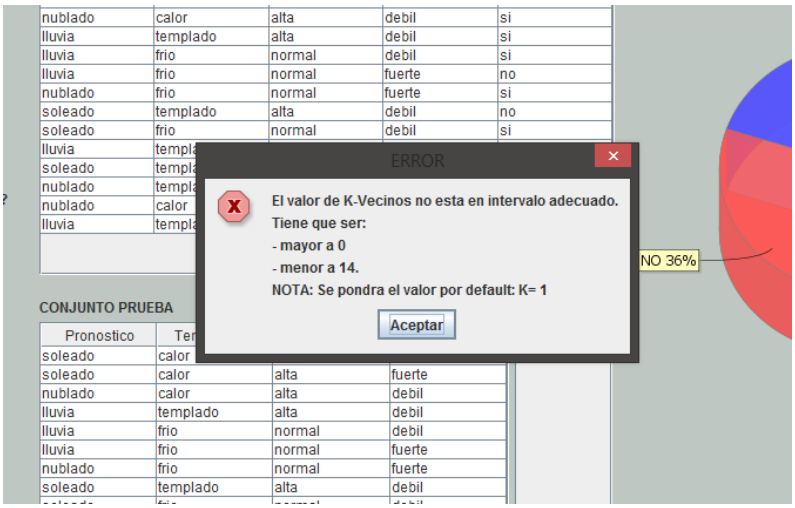



**Figura 3.28 Control de k Vecinos.**

El método **clasificaInstancia(instancia)** es el método principal que realiza las tareas importantes que distingue al algoritmo, este método realiza la invocación a métodos como CalculaDistancia(), Quicksort() y votación(). Al final regresa la clasificación propuesta para la instancia que fue enviada como parámetro.

**Nivel Código**

```
public int clasificaInstancia(int[] instancia){
        //Calculamos las distancias.
        double [][] distancias = calculaDistancias(instancia);
        //Ordenamos las distancias.
        quickSort(distancias,0,numFilas-2);
        //Se Vota por una clase.
        int ganadora = votacion(distancias,kVecinos);
        return ganadora;
}
```

El método **CalculaDistancia()** es el método en el que se genera una matriz guardando todas las distancias que hay entre el conjunto entrenamiento y la instancia prueba.

**Nivel Código**

```
private double [][] calculaDistancias(int[] instancia){
        int resta=0;
        double radicando;
        //distancia[0][...]: distancia de la instancia prueba.
        //distancia[1][...]: valor de clase de la instancia.
        double [][] distancia = new double [2][numFilas-1];
        for (int fila = 1; fila < numFilas; fila++){
                radicando = 0.0;
                for (int columna = 0; columna < columnaClase; columna++) {
                        resta = instancia[columna] - entrenamiento[fila][columna];
                        radicando += Math.pow(resta, 2);
                }
                distancia[0][fila-1] = Math.sqrt(radicando);
                distancia[1][fila-1] = entrenamiento[fila][columnaClase];
        }
        return distancia;
}
```

Puede resultar extraño guardar las distancias en una matriz, ya que solo se necesitaría un arreglo sin embargo tenemos que guardar las distancias en una matriz para no perder la clase a la que pertenece la distancia del ejemplo de entrenamiento a la instancia prueba.

Los for anidados sirven para recorrer el conjunto entrenamiento.

Para calcular las distancias (Véase Sección 2) primero tenemos que restar el atributo de la instancia de prueba con su correspondiente atributo del ejemplo de entrenamiento.

`resta = instancia[columna] - entrenamiento[fila][columna];`

Después se eleva al cuadrado para quitar los negativos `radicando += Math.pow(resta, 2);`

Después al resultado se saca raíz cuadrada, para finalmente ser guardada en la primera columna de la matriz distancia.

```
distancia[0][fila-1] = Math.sqrt(radicando);
```

Para no perder la clase a la que pertenece la distancia , se guarda en la segunda columna para posteriormente utilizarla para el criterio de votación.

El método **Quicksort()** se utilizó para ordenar las distancias de menor a mayor, con ese orden sería más fácil seleccionar los primeros k vecinos, ya que se buscan los k Vecinos más cercanos, como la estructura que hay que ordenar es una matriz se tuvo que ordenar con respecto a la columna distancias sin perder a su pareja la clase.

El método **votación()** se encargara de realizar un criterio de los Kvecinos más cercano, en este caso el criterio de votación es "Mayoría de clase", es decir, en la votación se hace un conteo de cuantas clases hay, la clase que tenga mayor número es la que gana. Por lo tanto esta clase es la que se propone y se regresa como clasificación de la instancia prueba.

**Nivel Código**

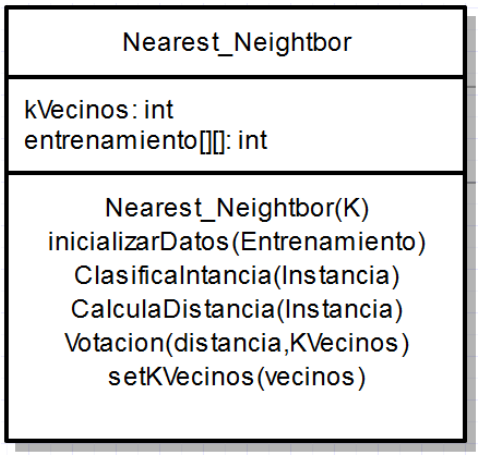
```
private int votacion(double[][] distancia, int kVecinos){
        int conteo;
        double valorClase;
        //Valores minimos
        int conteMayor = 0;
        double claseMayor = -1.0;

        for (int i = 0; i < kVecinos; i++) {
                conteo = 0;
                valorClase = distancia[1][i];
                if(valorClase == -1.0) continue;
                for (int j = i; j < kVecinos; j++)
                        if (valorClase == distancia[1][j]) {
                                conteo ++;
                                distancia[1][j] = -1;
                        }

                if (conteMayor <= conteo) {
                        claseMayor = valorClase;
                        conteMayor = conteo;
                }
        }
        return (int)claseMayor;
}
```

Llegamos a la parte final donde se decide que clase es la que se propone para la instancia prueba, en el código se propuso un algoritmo donde te diera la clase con mayor número de votos o bien la clase que más predominara.

Como pueden notar la instrucción distancia[1][j] = -1; modifica la matriz distancia, como nota la matriz distancia se pierde.

Como vimos la implementación de KNN fue simple, no se ocuparon reglas complejas fue simplemente aritmética euclidiana. No se realizó la división entre la fase de entrenamiento y la fase de clasificación, y en la implementación se llevó a cabo con un solo método.

Mostraremos el diagrama de clases de K vecinos más cercanos, para tener más claro su comportamiento.



## 3.3.4. Implementación de Iterative Dichotomiser

ID3 es un algoritmo recursivo, se trabaja con árboles de decisión, de los 3 clasificadores vistos hasta ahora ID3 es un poco más laborioso de implementar. En esta sección se explicara el algoritmo ID3, las estrategias de programación y detalles sobre cómo se llevó a cabo.

Al igual que Bayes ingenuo, ID3 es un algoritmo de clasificación que se divide en 2 fases, en la fase de entrenamiento y la fase de clasificación.

El algoritmo ID3 consta de un solo método el cual se llama así mismo recursivamente, las partes más importantes del algoritmo ID3 no fueron tratados en método diferentes, esto hace que el algoritmo ID3 sea un método muy largo pero más entendible, ya que si se realizaba de manera modular las cabecera de los métodos llegaban 6 parámetros haciendo que la cabecera sea muy larga, haciendo menos comprensible el código.

**Fase de construcción**

En esta fase se busca crear un de árbol mínimo de decisión para poder clasificar el conjunto prueba, A continuación una pequeña figura que muestra como un método llama a un método recursivo.

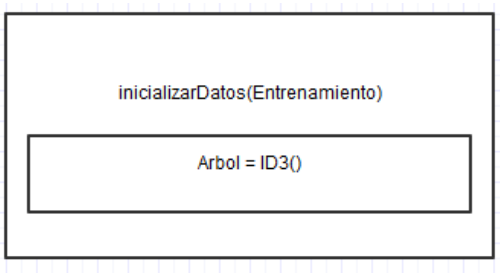



**Figura 3.29 fase de construcción**

**Nivel código**

```
public void inicializarDatos(int[][] entrenamiento) {
        arbol = iD3(entrenamiento);
}
```

La fase de construcción solo consta de un método, **InicializarDatos(Entrenamiento)** es un método que se encarga de crear el árbol de decisión, a diferencia de los demás clasificadores InicializarDatos(Entrenamiento) no inicializa variables como numclases, o numColumnas porque estas siempre van cambiando conforme la llamada recursiva vaya avanzando.

El método **ID3()** es el método que genera el árbol mínimo de decisión y lo guarda en una variable árbol para que posteriormente se ocupe para la fase de clasificación, aquí es en donde se generan las particiones , los nodos, Se calcula la cantidad de información, ganancia de información, etc.

A continuación se muestra una Figura mostrando el algoritmo ID3, dicho algoritmo representa de manera general el algoritmo ID3.

```
func ID3(X;C;A) _ ( X: Ejemplos, C: Clasificación, A: Atributos)
        si todos los ejemplos son de la misma clase
        entonces ID3   hoja con la clase.
        otro
        Calcular la función de cantidad de información de los ejemplos (I)
        para cada atributo en A
                Calcular la función de entropía (E) y la ganancia de información (G)
                Escoger el atributo que maximiza (G) (sea a)
                Eliminar a de la lista de atributos (A)
        termina
        para cada partición generada por los valores vi del atributo a
                Arboli   ID3(ejemplos de X con a = vi,Clasificación de los ejemplos, Atributos restantes)
                Generar árbol con a = viy Arboli
        termina
        ID3   la unión de todos los árboles

Termina ID3
```

**Figura 3.30 Algoritmo de ID3**

**Nivel Código**

```
private Nodo iD3(int[][] entrenamiento){
                //Datos
                int numFilas= entrenamiento.length;
                int numColumnas= entrenamiento[0].length;
                int columnaClase= numColumnas-1;

                if(sonClasesIguales(entrenamiento)){
                        //Variable que indica el indice del atributo clase.
                        int dato = getClasePromedio(entrenamiento,numFilas,columnaClase);
                        //Se crea el nodo clase
                        Nodo nodoClase = new Nodo();
                        nodoClase.setDato(dato);
                        nodoClase.setTipo("Clase");
                        return nodoClase;
                }else{
                //Arreglo que guardara el numero de elementos distintos de cada columna.
                        int [] numElemDist = new int [numColumnas];
                //Arreglo de listas que guardara todos los valores distintos de cada columna.
                        ArrayList [] valElemDist = new ArrayList[numColumnas];
                //Obtenemos los elementos diferentes de cada atributo.
                        for (int columna = 0; columna < numColumnas; columna++){
                           valElemDist[columna] = dameValorDist(entrenamiento,columna);
                           numElemDist[columna] = valElemDist[columna].size();
                        }
```

```
//Obtenemos el numero de clases
        int numClases= numElemDist[columnaClase];
        //HashMap que representara el modelo probabilistico.
        HashMap [][] modelo = new HashMap [numClases][numColumnas];

        //Calculamos el conteo de elementos.
        conteo(entrenamiento,valElemDist,numElemDist,modelo);

        //Calculamos la Cantidad de informacion
        double cantidadInfo = 0.0;
        cantidadInfo = numFilas * log2(numFilas);
        for (int clase = 0; clase < numClases ; clase++)
                for (int k = 0; k < numElemDist[columnaClase]; k++) {
                        int valor = (int) valElemDist[columnaClase].get(k);
                        int conteo = (int)
                        modelo[clase][columnaClase].get(valor);
                        cantidadInfo -= conteo*log2(conteo);
                }
        cantidadInfo /= numFilas;

        //Calculamos la Entropia de los atributos
        double [] entropias = new double[numColumnas-1];

        for (int columna = 0; columna < columnaClase; columna++) {
                entropias [columna] = 0;
                for (int k = 0; k < numElemDist[columna]; k++) {
                        int sumaValores = 0;
                        for (int clase = 0; clase < numClases ; clase++){

                                //Tomamos un valor distinto del atributo.
                                int valor = (int) valElemDist[columna].get(k);
                                int conteo = (int)
                                modelo[clase][columna].get(valor);
                                entropias [columna] -= conteo*log2(conteo);
                                sumaValores += conteo;
                        }
                    entropias [columna] += sumaValores*log2(sumaValores);
                }
                entropias [columna] /= numFilas;
        }

        //Calculamos la Ganancia de informacion
        int tamañoColum = entropias.length;
        double [] ganancias = new double[tamañoColum];
        for (int i = 0; i < ganancias.length; i++){
                ganancias[i] = cantidadInfo - entropias[i];
        }
```

```
        //buscamos la clase con mayor ganancia
                int atributoEliminar= 0;
                double mayor= ganancias[0];
                for (int i = 1; i < ganancias.length; i++){
                        if(mayor < ganancias[i]){
                                atributoEliminar= i;
                                mayor= ganancias[i];
                        }
                }
                //Informacion del atributo a eliminar
                int dato = entrenamiento[0][atributoEliminar];
                String tipo = "Atributo";
                ArrayList<Integer> hijos = valElemDist[atributoEliminar];
                //Se crea el nodo en el arbol
                Nodo atributo = new Nodo();
                atributo.setDato(dato);
                atributo.setTipo(tipo);
                atributo.setHijos(hijos,"Valor");
                //se crean las particiones del atributo
                for (int i = 0; i < numElemDist[atributoEliminar]; i++) {
                        int valor = (int) valElemDist[atributoEliminar].get(i);
                //En la particion solo que consideran las filas con el valor.
                        int [][] particion=
                        crearPartion(entrenamiento,valor,atributoEliminar);
                        //se borra el atributo de la particion.
                        particion = borrarColumna(particion,atributoEliminar);
                        //Se genera la recursividad, se realiza lo mismo para la
                        particion.
                        Nodo subArbol = iD3(particion);
                        //Pegamos el subArbol al nodo atributo
                        atributo.getHijos().get(i).setHijo(subArbol);
                }
                //juntamos los arboles
                return atributo;
        }
}
```

El código de la función ID3 recursivo es muy extenso, A continuación se explicara por medio de bloques importantes, siempre nos guiaremos y compararemos con el algoritmo de la figura 3.30.

Detalles del código

```
//Datos
        int numFilas= entrenamiento.length;
        int numColumnas= entrenamiento[0].length;
        int columnaClase= numColumnas-1;
```

Al principio se sacan los datos iniciales, estos datos serán necesarios más adelante en el código, podemos notar que estos datos iniciales cambiaran conforme se haga la recursividad porque las particiones se harán más pequeñas.

```
if(sonClasesIguales(entrenamiento)){
        //Variable que indica el indice del atributo clase.
        int dato = getClasePromedio(entrenamiento,numFilas,columnaClase);
        //Se crea el nodo clase
        Nodo nodoClase = new Nodo();
        nodoClase.setDato(dato);
        nodoClase.setTipo("Clase");
        return nodoClase;
}
```

El if anterior es la condición de paro, este if no permite que se cicle el algoritmo.
El método de paro sonClasesIguales(entrenamiento) del algoritmo recursivo tiene el siguiente comportamiento:
Si **true** entonces pueden suceder dos cosas en la partición solo quedan instancias de la misma clase, esto implica detener la recursividad, creamos la hoja con la clase común. O ya no hay instancias en la partición pero si hay diferentes clases, para solucionar esta situación en donde siempre habrá un error, se sacara el promedio y se sacara la clase más común para crear la hoja.

```
//Arreglo que guardara el numero de elementos distintos de cada columna.
        int [] numElemDist = new int [numColumnas];
//Arreglo de listas que guardara todos los valores distintos de cada columna.
        ArrayList [] valElemDist = new ArrayList[numColumnas];
//Obtenemos los elementos diferentes de cada atributo.
        for (int columna = 0; columna < numColumnas; columna++){
                valElemDist[columna] = dameValorDist(entrenamiento,columna);
                numElemDist[columna] = valElemDist[columna].size();
        }
        //Obtenemos el numero de clases
        int numClases= numElemDist[columnaClase];
        //HashMap que representara el modelo probabilistico.
```

```
HashMap [][] modelo = new HashMap [numClases][numColumnas];

//Calculamos el conteo de elementos.
conteo(entrenamiento,valElemDist,numElemDist,modelo);
```

Este bloque de código realiza las preparaciones de datos para los cálculos, podemos observar que tan parecido es este bloque con algunas líneas de código de bayes ingenuo, aquí también se sacan los valores diferentes de los atributos, con estos datos podemos crear el modelo que guardara el conteo, para realizar más fácil los cálculos de las probabilidades, las cuales se ocuparan en la entropía de cada atributo.

```
//Calculamos la Cantidad de informacion
                    double cantidadInfo = 0.0;
                    cantidadInfo = numFilas * log2(numFilas);
                    for (int clase = 0; clase < numClases ; clase++)
                            for (int k = 0; k < numElemDist[columnaClase]; k++) {
                                    int valor = (int) valElemDist[columnaClase].get(k);
                                    int conteo = (int)
                                    modelo[clase][columnaClase].get(valor);
                                    cantidadInfo -= conteo*log2(conteo);
                            }
                    cantidadInfo /= numFilas;
```

Como indica el algoritmo lo primero que se realiza es calcular la cantidad de información, aquí los datos que se tienen que calcular son las probabilidades a priori.

Podemos notar en las líneas de código que se maneja el logaritmo base 2, este método se implementó para poder obtener logaritmos en base 2, otro de los objetivos fue controlar cuando en el argumento de los logaritmos sea cero.

La cantidad de información se guarda en una variable cantidadInfo;

```
//Calculamos la Entropia de los atributos
                    double [] entropias = new double[numColumnas-1];

                    for (int columna = 0; columna < columnaClase; columna++) {
                            entropias [columna] = 0;
                            for (int k = 0; k < numElemDist[columna]; k++) {
                                    int sumaValores = 0;
                                    for (int clase = 0; clase < numClases ; clase++){

                                            //Tomamos un valor distinto del atributo.
                                            int valor = (int) valElemDist[columna].get(k);
                                            int conteo = (int)
                                            modelo[clase][columna].get(valor);
```

```
                    entropias [columna] -= conteo*log2(conteo);
                    sumaValores += conteo;
                }
            entropias [columna] += sumaValores*log2(sumaValores);
        }
        entropias [columna] /= numFilas;
    }
```

Este bloque de código calcula la entropía de cada atributo del conjunto entrenamiento, las probabilidades que ahora se deben sacar son las condicionales. Las entropías se guardan en un arreglo de tamaño del número de los atributos.

```
//Calculamos la Ganancia de informacion
        int tamañoColum = entropias.length;
        double [] ganancias = new double[tamañoColum];
        for (int i = 0; i < ganancias.length; i++){
            ganancias[i] = cantidadInfo - entropias[i];
        }
```

Ya con las entropías se calcula la ganancia de información, como la teoría lo indica solo se realiza la resta entre la cantidad de información y la entropía.
ganancias[i] = cantidadInfo - entropias[i];
Estas ganancias de información se guardan en otro arreglo.

```
//buscamos la clase con mayor ganancia
        int atributoEliminar= 0;
        double mayor= ganancias[0];
        for (int i = 1; i < ganancias.length; i++){
            if(mayor < ganancias[i]){
                atributoEliminar= i;
                mayor= ganancias[i];
            }
        }
```

Este bloque de código busca la mayor ganancia de información de un atributo, se guarda el índice del atributo que tiene mayor ganancia de información.

```
//Informacion del atributo a eliminar
                    int dato = entrenamiento[0][atributoEliminar];
                    String tipo = "Atributo";
                    ArrayList<Integer> hijos = valElemDist[atributoEliminar];
                    //Se crea el nodo en el arbol
                    Nodo atributo = new Nodo();
                    atributo.setDato(dato);
                    atributo.setTipo(tipo);
                    atributo.setHijos(hijos,"Valor");
```

El atributo que tiene mayor ganancia de información será eliminado de la tabla y se generaran las particiones correspondientes, ya que tenemos el atributo que será agregado al árbol de decisión entonces es prudente el realizar el nodo con sus respectivos hijo.

Este bloque de código crea el nodo con sus respectivos hijos, aunque todavía no unimos.

```
        //se crean las particiones del atributo
                    for (int i = 0; i < numElemDist[atributoEliminar]; i++) {
                            int valor = (int) valElemDist[atributoEliminar].get(i);
                    //En la particion solo que consideran las filas con el valor.
                            int [][] particion=
                            crearPartion(entrenamiento,valor,atributoEliminar);
                            //se borra el atributo de la particion.
                            particion = borrarColumna(particion,atributoEliminar);
```

Ahora resta realizar las particiones correspondientes del atributo a eliminar, la forma de realizarlo es por medio de un método crearParticion() en el cual tiene como parámetros de entrada el entrenamiento, el valor que se tiene que mantener y el atributo a eliminar, este método regresara una matriz que es en realizad una partición de "valor". Después de generar la partición se borra la columna del atributo con mayor ganancia, una vez más se creó un método que nos ayudara realizar esta tarea
borrarColumna(particion,atributoEliminar);
Este método es muy simple y su única función es el de eliminar una columna, regresa otra partición pero ahora sin la columna del atributo con mayor ganancia.

```
                    //Se genera la recursividad, se realiza lo mismo para la partición.
                    Nodo subArbol = iD3(particion);
                    //Pegamos el subArbol al nodo atributo
                    atributo.getHijos().get(i).setHijo(subArbol);
            }
```

En este bloque de código se genera la recursividad, el método iD3 se llama a si mismo pero con una diferencia, ahora le pasamos la partición creada y se vuelve a hacer el mismo proceso,

tenemos que tener en cuenta que iD3 regresa un árbol, por lo tanto cuando recibamos el árbol de iD3 tenemos que guardar el subárbol.
La línea que sigue se encarga de pegar el árbol con el subárbol en la rama correspondiente.
atributo.getHijos().get(i).setHijo(subArbol);
Anteriormente creamos el nodo que representara el atributo en el árbol de decisión.

```
//juntamos los arboles
return atributo;
```

La última línea retorna el nodo atributo que será pegado en la anterior rama.

La fase de construcción se realizó en un solo método, el cual regreso un árbol de decisión.

Algunos métodos que son importantes y que fueron pieza fundamental son los siguientes:

**Son clases iguales**

```
//Condicion de parada para el metodo recursivo
private boolean sonClasesIguales(int[][] subEntrenamiento) {
        int numFilas= subEntrenamiento.length;
        int columnaClase= subEntrenamiento[0].length-1;
        //tomamos el primer valor para compararlo con todas las clases restantes
        int tomaClase = subEntrenamiento[1][columnaClase];
        for (int fila = 2; fila < numFilas; fila++){
        if(tomaClase != subEntrenamiento[fila][columnaClase]
                                        && columnaClase != 0)
                return false;
        }
        return true;
}
```

La idea es tomar la primera clase y compararla con las demás, en dado caso que al recorrer todas y estoy comparando con la última clase entonces devolvemos true y decimos que son iguales todas, en caso contrario saldremos del método más rápido regresando un false.

**Crear partición**

```
private int[][] crearPartion(int[][] entrenamiento, int valor, int atributoEliminar) {
        int numfilasValor=0;
        int numFilas= entrenamiento.length;
        int numColumnas= entrenamiento[0].length;

        for (int i = 0; i < numFilas; i++)
                if(entrenamiento[i][atributoEliminar] == valor)
                        numfilasValor++;
```

```
        //Encabezados por eso es el +1
        int [][] particion = new int [++numfilasValor][numColumnas];
        //copiamos los encabezados
        particion[0]= entrenamiento[0];

        for (int i = 1; i < numFilas ; i++){
                if(entrenamiento[i][atributoEliminar] == valor)
                        particion[--numfilasValor]= entrenamiento[i];
        }
        return particion;
}
```

El primer for es para recorrer todo la columna del atributo para saber cuantas filas son de la partición del atributo valor. El primer if es para comparar si la fila actual la tenemos que considerar dentro de la partición.

La estrategia que se utilizo fue generar otra matriz y recorrer con ayuda del número de filas de la partición del primer for, y pegar los ejemplo completos a la matriz particion.

**Borrar columna**

```
private int[][] borrarColumna(int[][] entrenamiento, int atributoEliminar){
        int numFilas= entrenamiento.length;
        int numColumnas= entrenamiento[0].length;
        int [][] subEntrenamiento = new int [numFilas][numColumnas-1];

        for (int fila = 0; fila < numFilas; fila++) {
                for (int columna = 0; columna < numColumnas; columna++) {
                        if(columna < atributoEliminar)
                                subEntrenamiento[fila][columna] =
                                entrenamiento[fila][columna];
                        else{
                                if(columna>atributoEliminar){
                                        subEntrenamiento[fila][columna-1] =
                                        entrenamiento[fila][columna];
                                }
                        }
                }
        }
        return subEntrenamiento;
}
```

Se crea otra matriz pero con una columna menos, la estrategia que se siguió es simple copias la partición en otra, pero no se considera la columna a eliminar eso se puede observar en las dos sentencias if, que controlan el acceso por el lado izquierdo y derecho de la columna.

**Fase de Clasificación**

La fase de clasificación ocupa el árbol de decisión creado en la en la fase de construcción, la manera en que la fase de clasificación ocupa el árbol es recorriendo el árbol por cada instancia introducida, es decir, se recorrerá el árbol desde la raíz siguiendo las ramas según el valor de sus atributo de la instancia prueba, a continuación se muestra un pequeño diagrama del funcionamiento de la fase de Clasificación.

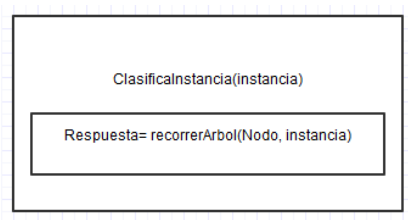



**Figura 3.31 Fase de Clasificación**

El método **ClasificaInstancia(instancia)** se encarga de proponer una clase a la instancia de prueba, para ello se invoca otro método llamado recorrerArbol(Nodo,instancia) el cual se encarga de buscar la clase propuesta.

**Nivel Código:**

```
public int clasificaInstancia(int[] intancia) {
        int respuesta = recorrerArbol(arbol,intancia);
        return respuesta;
}
```

**RecorrerArbol(Nodo, instancia)** es un método recursivo, la tarea principal de recorrer el árbol y buscar la clase propuesta no es una tarea muy difícil de realizar, pero puede generar un problema cuando hay valores de atributos desconocidos, cuando le introducen una instancia de prueba con valores que no están registrados en el árbol este arroja un -1, esto implica que no encontró una clase para la instancia prueba y el algoritmo pasa por alto esa instancia no clasificándola con ninguna clase.
Cuando la instancia la clasifico correctamente la regresa, y espera a que le den otra.

**Nivel Código**

```
private int recorrerArbol(Nodo nodo, int[] intancia){
                int numHijos = nodo.getHijos().size();
                if( numHijos > 0){
                        if(nodo.getTipo().equals("Atributo")){
                                int indice = buscaIndice(nodo.getDato());
                                int valor = intancia[indice];
                                for (int i = 0; i < numHijos ; i++)
                                        if(nodo.getHijos().get(i).getDato() == valor)
                                                return
                                recorrerArbol(nodo.getHijos().get(i),intancia);
                        }else
                                if(nodo.getTipo().equals("Valor"))
                                        return recorrerArbol(nodo.getHijos().get(0),intancia);
```

```
        }else
                if(nodo.getTipo().equals("Clase")){
                        return nodo.getDato();
                }
        return -1;
}
```

El árbol final tiene 3 tipos de nodo;

- Nodo “Clase”
- Nodo “Atributo”
- Nodo “Valor”

La primera instrucción es comprobar si tengo nodos hijo, **if**( numHijos > 0)
Ya que si resulta que no tengo nodos hijos quiere decir que ya llegue a algún nodo hoja, por lo tanto vamos enviar el valor del nodo hoja que es la clase que ID3 propone como la clasificación de la instancia prueba.

```
if(nodo.getTipo().equals("Clase")){
        return nodo.getDato();
}
```

En dado caso que tenga nodos hijo entonces debemos de comprobar qué clase de nodo son

```
if(nodo.getTipo().equals("Atributo")){
        //código omitido;
}else
        if(nodo.getTipo().equals("Valor"))
        return recorrerArbol(nodo.getHijos().get(0),intancia);
```

Y decidir para que lado me voy, si el nodo es de tipo “Valor” entonces de forma recursiva nos movemos hacia esa rama sin importar nada ya que solo es un valor, pero Si el nodo es de tipo “Atributo” entonces debemos de saber por qué lado hay que seguir ya que el atributo tiene muchos nodos hijo que son los valores de los atributos.

```
int indice = buscaIndice(nodo.getDato());
int valor = intancia[indice];
```

Ahora ya que sabemos por dónde nos debemos de mover, proseguimos a movernos de manera recursiva, recorriendo los hijos para buscar la rama correcta.

```
for (int i = 0; i < numHijos ; i++)
        if(nodo.getHijos().get(i).getDato() == valor)
                return recorrerArbol(nodo.getHijos().get(i),intancia);
```

Al final recorrer y encontrar una clase que ID3 proponga para la instancia prueba, se regresa recursivamente hasta que llega al nodo raíz y este la guarda en una variable como respuesta.

Esta implementación de ID3 puede resultar no modular, pero si resulta más fácil seguir el código y no perdernos con las recursividades. Tampoco es muy costoso en cómputo como lo es KNN, y más realista que Bayes ingenuo. ID3 es uno de los más populares y más utilizados en el aprendizaje supervisado.

# Experimentos y resultados

## 4.1. Base de Datos utilizada

Como mencionamos en la sección 2.4 contábamos con una Base de Datos que se encontraba en un estado crudo, la cual fue sometida al proceso KDD obteniendo un formato de tabla, el cual es más fácil de utilizar por cada uno de los clasificadores implementados, esta base tiene la estructura que se muestra en la **Tabla 4.1**

| TempReaccion | TiempInyec | TiempCata | TiempLiq | RelacionCO | Clase |
|---|---|---|---|---|---|
| 520 | 120 | 350 | 840 | 3.75 | media |
| 520 | 45 | 305 | 315 | 10 | buena |
| 535 | 120 | 350 | 840 | 3.75 | media |
| 535 | 45 | 305 | 315 | 10 | buena |
| 550 | 120 | 350 | 840 | 3.75 | media |
| 550 | 45 | 305 | 315 | 10 | buena |
| 565 | 120 | 350 | 840 | 3.75 | buena |
| 565 | 45 | 305 | 315 | 10 | buena |
| 520 | 30 | 230 | 240 | 3 | mala |
| | | … | | | |
| 480 | 60 | 350 | 420 | 7.5 | media |

**Tabla 4.1 Base de Datos principal**

En donde los atributos son: TempReaccion, TiempInyec, TiempCata, TiempLiq RelacionCO y el atributo que seleccionamos como la clase.

La principal base obtenida por el proceso KDD de un Reactor de Microactividad de Cracking Catalítico consta de 321 ejemplos, la cual es una base aun no explorada en su totalidad y tiene una distribución como se muestra en la gráfica de pastel de la **Figura 4.2.**

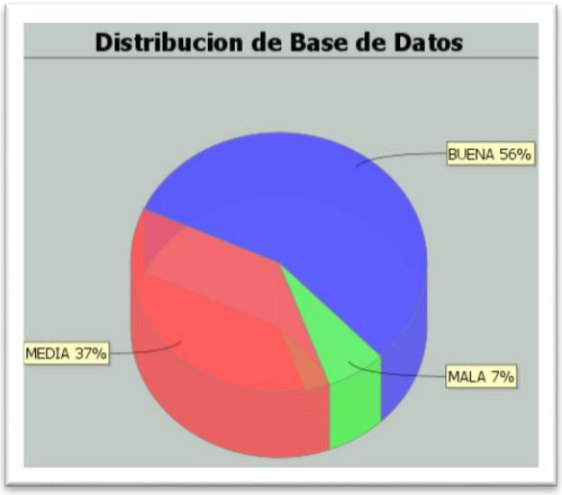



**Figura 4.2 Distribución de Base de Datos**

Para obtener más datos de prueba se planteó la posibilidad de combinar los datos que teníamos a fin de obtener un conjunto más grande de la base inicial.

La base que obtuvimos mediante la combinación de los atributos con cada clase nos dio un total de 76, 008 ejemplos, la cual tiene una distribución de la forma que se muestra en la **Figura 4.3** y esta es la que se utilizó para hacer la clasificación con los diferentes modelos de clasificación.

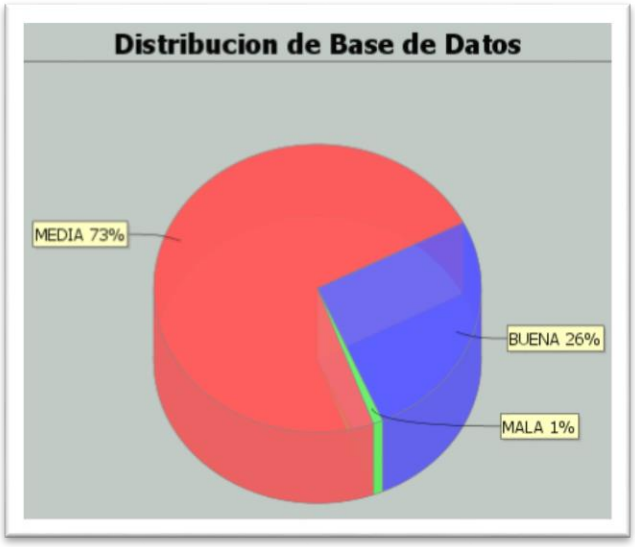



**Figura 4.3 Distribución de Base de Datos utilizada**

La distribución que se puede observar es el 26% para la clase BUENA, 73% para la clase MEDIA y 1% para la clase MALA, lo cual es una distribución que pudiera hacer que tengamos un problema de generalización.

Por ejemplo si obtenemos buenos resultados al clasificar, un 80% de la medida de exactitud esta podría ser porque clasifico la mayoría como de clase MEDIA y unos de la clase BUENA, teniendo un problema ya que la clase MALA no sería contemplada y esto aunque nos da una medida de exactitud alta, aun así no sería una buena clasificación.

4.2. Planteamiento de experimentos

En esta sección se mostraran los experimentos que se llevaron a cabo en este Proyecto Terminal. Estos experimentos se realizaron mediante diferentes métodos como son Método Tradicional, Método Validación Cruzada y Partición por Porcentaje, los cuales fueron probados mediantes los distintos clasificadores.

Con estos experimentos se logrará ver cuál de todos los métodos y clasificadores, fueron los que nos arrojaron mejores resultados.

Las características del hardware de la computadora utilizada para hacer las pruebas son las siguientes:

- Procesador: ADM A8-5545M APU with Radeon™ HD Graphics 1.70 GHz
- Memoria RAM: 8.00 GB (7.19 GB utilizable)
- Tipo de sistema: Sistema operativo de 64 bits, procesador x64

# 4.2.1. Métodos de clasificación

Con métodos de clasificación nos referimos a la forma en que fue tomado el conjunto prueba, por ejemplo este pudiera ser el mismo que el conjunto entrenamiento, o solo ser una porción de este.

Se implementaron tres distintas formas de clasificación, que son: Método tradicional, Método de Validación Cruzada y Partición por Porcentaje, con los cuales se harán pruebas en la siguiente sección.

Obtenemos dos resultados al clasificar, los cuales son el tiempo en que se tardó en clasificar y la medida de funcionamiento exactitud, con lo cual se dividirán en dos secciones de resultados. Y al finalizar podremos comparar cuál de los tres métodos de clasificación nos dio mejores resultados, ya sea en tiempo o en la medida de funcionamiento exactitud.

## 4.2.1.1. Método Tradicional analizando la exactitud

Este método consiste en tomar el 100% de los elementos para el conjunto entrenamiento y el 100% de elementos para el conjunto prueba.

**Resultado de experimento con Bayes Ingenuo**

Después de realizar una serie de experimentos el resultado obtenido con el clasificador Bayes Ingenuo al utilizar el método tradicional se muestra en la **Tabla 4.2**

| Bayes Ingenuo con Método Tradicional | |
|---|---|
| Exactitud | 88.28% |

**Tabla 4.2 Método Tradicional con BI - Exactitud**

En el cual podemos ver que se obtuvo un 88.28% que es un muy buen resultado de clasificación.

**Resultados de experimentos con K-vecinos más cercanos**

Con K-vecinos más cercanos se realizaran varias pruebas modificando el número de vecinos. Los vecinos utilizados son: 1, 5, 10, 15 y 20. Con lo cual al final se realizó un promedio de los resultados obtenidos.

Los resultados obtenidos con el clasificador K-vecino más cercano al utilizar el método tradicional, se muestra en la **Tabla 4.3**

| K- vecinos más cercanos con Método Tradicional | |
|---|---|
| Exactitud | 86.008% |

**Tabla 4.3 Método Tradicional con KNN - Exactitud**

Como se observa en la Tabla 4.3 el clasificador k-vecinos más cercano obtuvo el 86.008% de exactitud, utilizando el método tradicional, lo cual es más bajo que lo obtenido con Bayes Ingenuo.

**Resultado de experimento con ID3**

El resultado obtenido con el clasificador ID3 al utilizar el método tradicional, se muestra en la **Tabla 4.4.**

| ID3 con Método Tradicional | |
|---|---|
| Exactitud | 88.28% |

**Tabla 4.4 Método Tradicional con ID3 - Exactitud**

Observando las Tablas 4.2, 4.3 y 4.4, podemos ver que los clasificadores que obtuvieron la medida de funcionamiento exactitud más alta son 2: Bayes Ingenuo e ID3, estos dos clasificadores obtuvieron la misma exactitud, aunque se realizaron varias veces el experimento los resultados siempre dieron 88.28% y esto se debe a que estos dos tipos de clasificadores arrojan la misma clasificación si se tiene como entrada el mismo conjunto de entrenamiento y el conjunto prueba.

El segundo método que se revisó fue el método de Validación Cruzada, en el cual ya no se va tener el 100% de elementos para el conjunto entrenamiento al igual que para el conjunto prueba, sino que ahora se va a dividir la Base de Datos en diferentes secciones las cuales van a ir cambiando para el conjunto entrenamiento como para el conjunto prueba.

### 4.2.1.2. Método Validación Cruzada analizando la exactitud

Cómo ya se explicó en la sección 4.1 el método de validación cruzada consiste en particionar la Base de Datos. Siendo el conjunto prueba una de estas particiones mientras que las otras son el conjunto entrenamiento. La partición que se da como el conjunto prueba se va modificando después de realizar la clasificación tantas particiones sean.

**Resultado de experimento con Bayes Ingenuo**

El primer clasificador que presentaremos es el de Bayes Ingenuo, en donde se puso a prueba con el método de validación cruzada obteniendo diverso resultados para las diferentes particiones las cuales se consideraron de 2, 4, 6, 8 y 10 . Los resultados obtenidos se presentan en la **Figura 4.4**

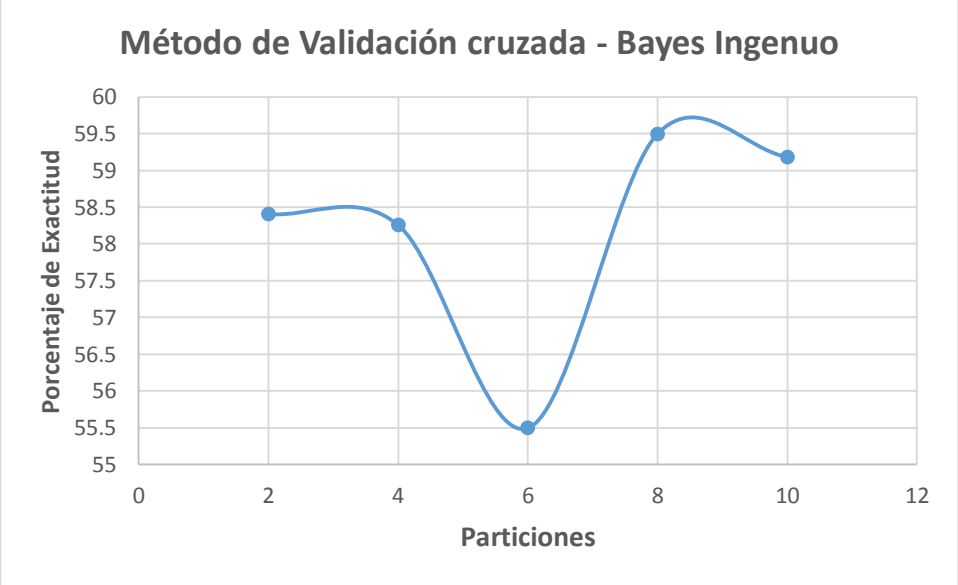



**Figura 4.4 Método Validación Cruzada-Exactitud-BI**

Se puede observar en la Figura 4.4, que la exactitud obtenida con Bayes Ingenuo utilizando el método de validación cruzada oscilan entre el 55.5% y el 59.5 %. Se obtuvieron dichos resultados después de que se realizaron 25 pruebas.

Como se observa los porcentajes de exactitud obtenidos son menores que con el método tradicional como se muestra en la **Figura 4.5**

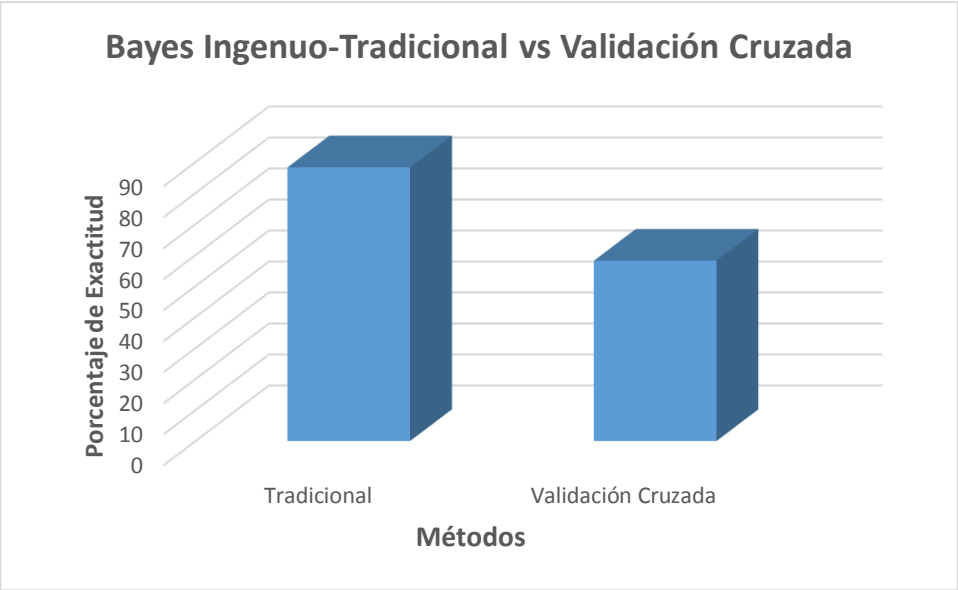



**Figura 4.5 Validación Cruzada vs Tradicional – BI - exactitud**

Esto se debe al conjunto entrenamiento y al conjunto prueba que se utilizaron. Ya que los elementos de estos son seleccionados aleatoriamente por lo cual en algún momento se pudo tener la mayoría de la clase Media y Buena para el conjunto entrenamiento y en el conjunto prueba tener más de la clase Baja, con lo cual al momento de clasificarlos con Bayes Ingenuo y no tener muchos ejemplos de la clase Baja, estos iban a ser clasificados con otra clase.

## Resultados de experimentos con K-vecinos más cercanos

Para K vecinos más cercanos utilizando el método de validación cruzada después de realizar 25 pruebas con los diferentes vecinos antes mencionados se obtuvieron los resultados que se muestran en la **Figura 4.6**

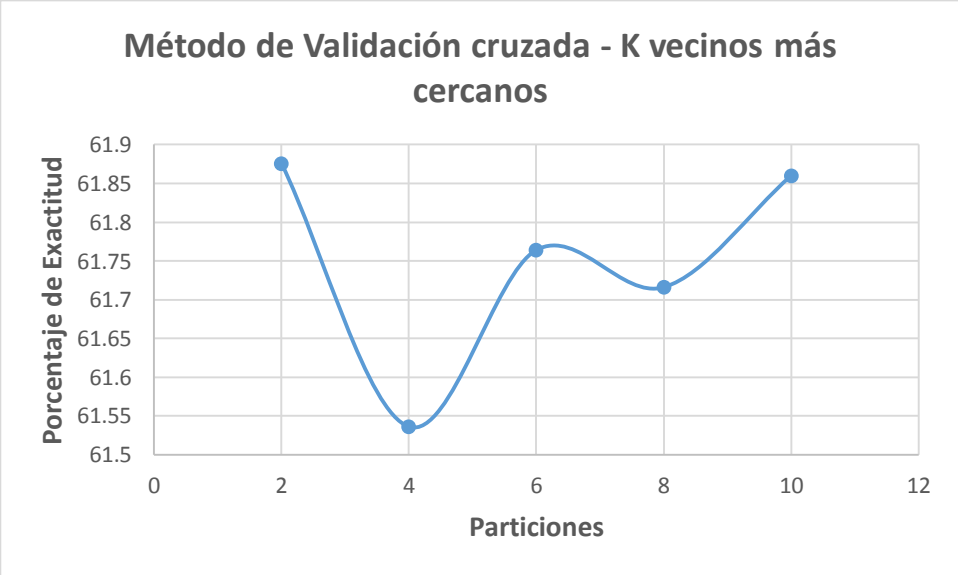



**Figura 4.6 Método Validación Cruzada-Exactitud-KNN**

Como se observan los porcentajes de exactitud que se obtuvieron oscilan entre el 61.53% y el 61.87%, lo cual también se debe por la forma en que fueron tomados el conjunto prueba y el conjunto entrenamiento. También varían los resultados porque se tomó el promedio de las 25 pruebas y como la clasificación que se obtiene con k vecinos más cercanos difiere dependiendo de los vecinos utilizados.

Con lo cual podemos ver que el porcentaje de exactitud obtenido con el método de validación Cruzada es menor que el obtenido con el método Tradicional como se muestra en la **Figura 4.7**

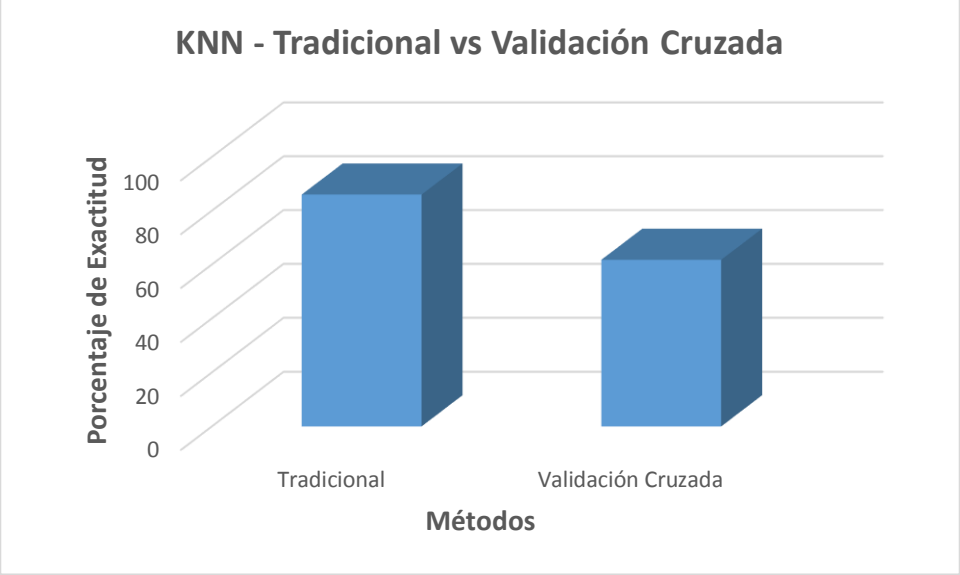



**Figura 4.7 Validación Cruzada vs Tradicional – KNN - exactitud**

**Resultado de experimento con ID3**

El tercer clasificador que se reviso fue ID3 utilizando el método de validación cruzada se realizaron 25 pruebas con las cuales se obtuvieron los resultados que se muestran la **Figura 4.8**

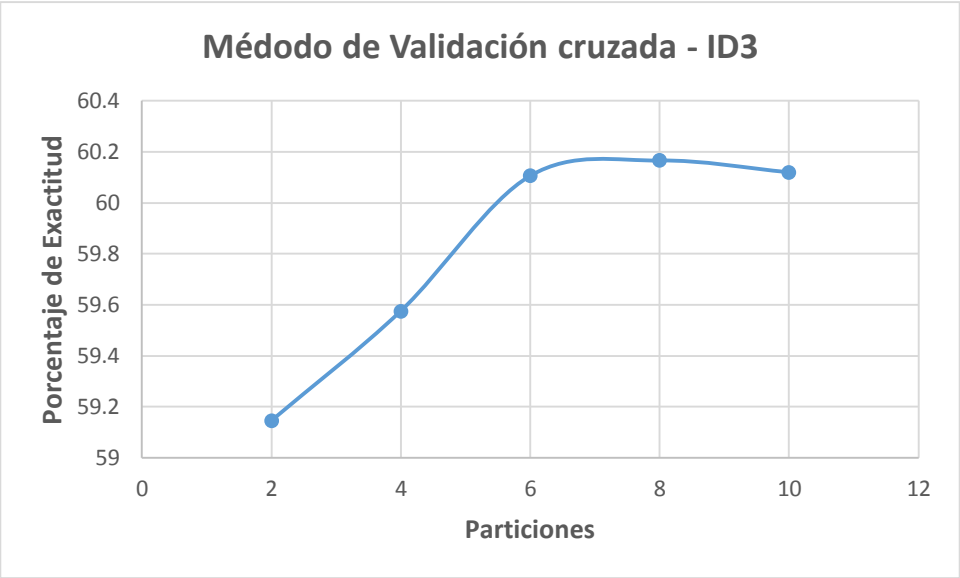



**Figura 4.8 Método Validación Cruzada-Exactitud-ID3**

Como se observa el porcentaje de exactitud en la figura 4.8 va en forma creciente, mediante el número de particiones crece el porcentaje de Exactitud también hasta que se estabiliza, obteniendo como el porcentaje de exactitud más alto de 60.16% pero sacando el promedio de las pruebas obtenemos un resultado de 59.82%, con lo cual tampoco es mejor que el resultado obtenido con el método tradicional como se observa en la **Figura 4.9**

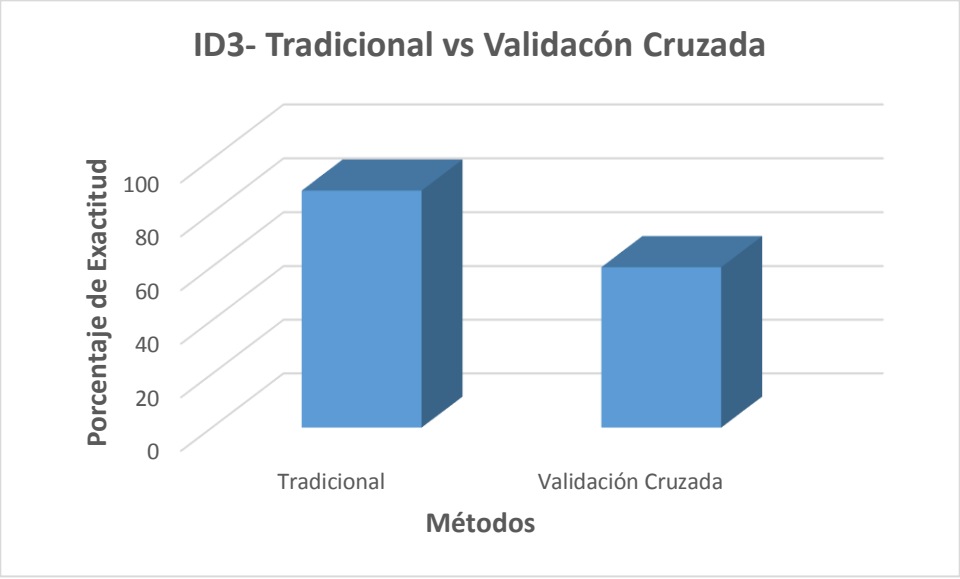



**Figura 4.9 Validación Cruzada vs Tradicional – ID3 - exactitud**

El método tradicional nos dio un 88.28% en el porcentaje de exactitud, lo cual es muy alto a comparación con el método de validación cruzada, esto se debe a la forma en cómo fueron tomados el conjunto entrenamiento y el conjunto prueba, ya que ID3 tiene problemas cuando en el conjunto entrenamiento no se tienen algunos ejemplos que están en el conjunto prueba

este no sabe cómo clasificarlos, por lo cual nos los clasifica y al no tener clase baja el porcentaje de exactitud.

### 4.2.1.3. Método Partición por porcentajes analizando la exactitud

Al utilizar este método se necesita indicar el porcentaje que requieres para el conjunto prueba. Con el porcentaje indicado toma los elementos aleatoriamente del conjunto entrenamiento, quitando los elementos del conjunto entrenamiento que fueron tomados para el conjunto prueba, siendo así que la Base de Datos inicial es dividida entre el conjunto entrenamiento y conjunto prueba respecto al porcentaje que se dio.

Los porcentajes que se utilizaron para realizar las pruebas con los diferentes clasificadores fueron 50%, 40%, 30%, 20% y 10%, con lo cual se hicieron un total de 25 pruebas con cada uno de los clasificadores.

**Resultado de experimento con Bayes Ingenuo**

Los resultados obtenidos para el clasificador Bayes Ingenuo utilizando el método de Partición por Porcentaje se muestran en la **Figura 4.10**

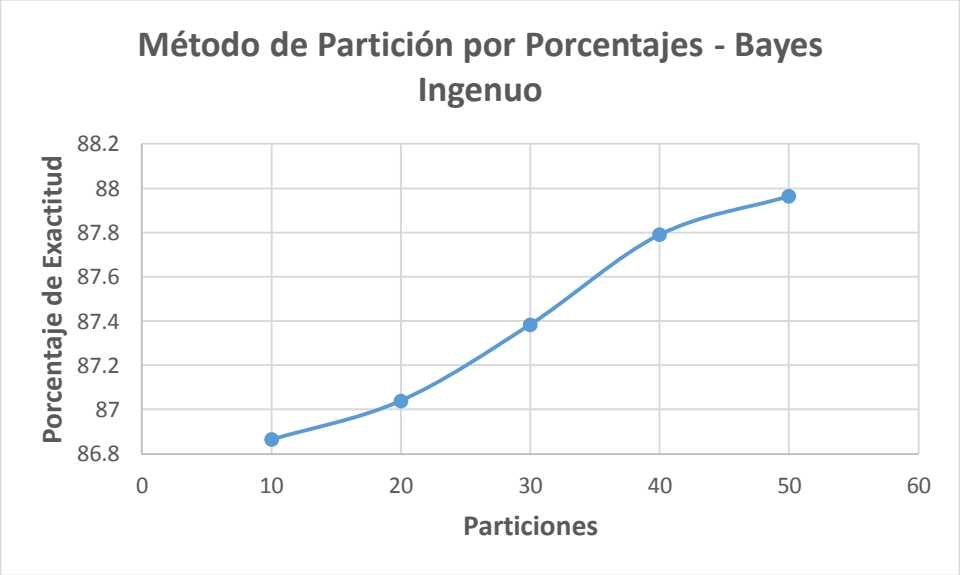



**Figura 4.10 Método Partición por Porcentaje - Exactitud - BI**

Con lo cual podemos observar que el porcentaje de exactitud va aumentando conforme le agregamos mayor cantidad del porcentaje de la Base de Datos al conjunto prueba. Sacando el promedio de estos porcentajes obtenemos un 87.40 el cual es muy bueno, aunque si lo comparamos con el método tradicional este es menor como se muestra en la **Figura 4.11**

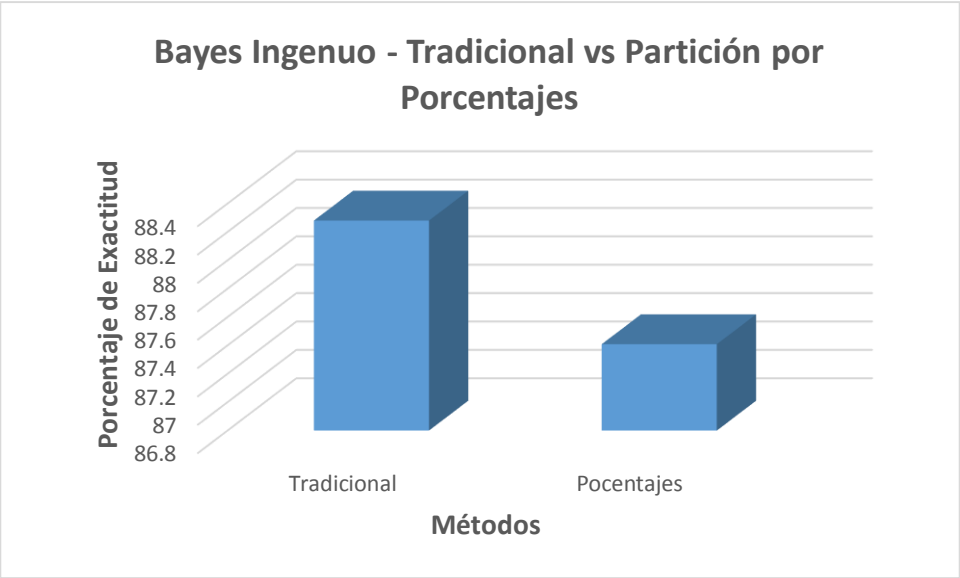



**Figura 4.11 Partición por Porcentaje vs Tradicional – BI - exactitud**

El porcentaje de exactitud obtenido con el método tradicional en Bayes Ingenuo fue del 88.28% y el obtenido con el método de Partición por Porcentajes fue del 87.40% que al igual los dos resultados obtenidos fueron muy buenos.

**Resultados de experimentos con K-vecinos más cercanos**

Los resultados obtenidos para el clasificador K vecinos más cercanos utilizando el método de Partición por Porcentaje se muestran en la **Figura 4.12**

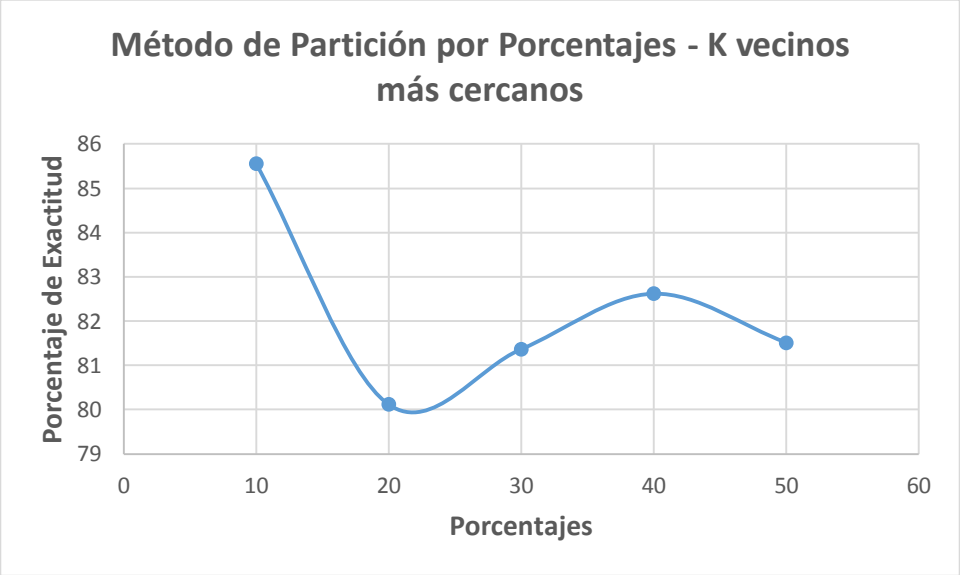



**Figura 4.12 Método Partición por Porcentaje - Exactitud - KNN**

Se observa que con el 10% utilizado en el conjunto prueba se obtuvo mayor porcentaje de exactitud, esto se debe a que se tiene mayor número de ejemplos en el conjunto entrenamiento con lo cual al clasificar los ejemplos del conjunto prueba se tienen más ejemplos cercanos al ejemplo que se está clasificando con lo que se hace una mejor clasificación.

Si comparamos el resultado obtenido con el método tradicional y el método de Partición por Porcentajes el método tradicional arroja mejores resultados en el porcentaje de exactitud como se muestra en la **Figura 4.13**

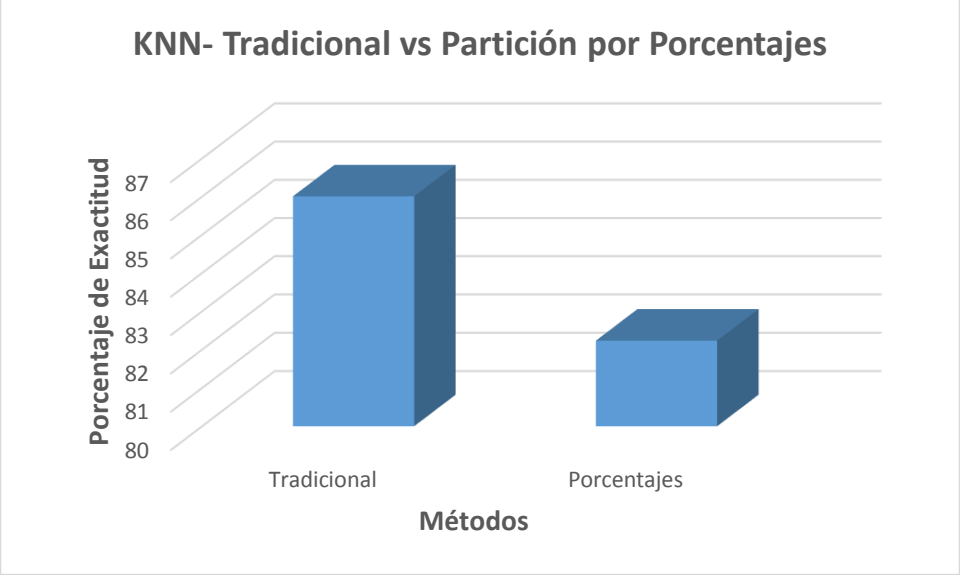



**Figura 4.13 Partición por Porcentaje vs Tradicional – KNN - exactitud**

Como se observa se obtuvo un porcentaje de exactitud del 86.00% con el método tradicional y un 83.23% con el método de Partición por Porcentajes, con lo cual los dos son muy buenos resultados, pero de igual forma sigue siendo mejor el método tradicional.

## Resultado de experimento con ID3

Los resultados obtenidos para el clasificador K vecinos más cercanos utilizando el método de Partición por Porcentaje se muestran en la **Figura 4.14**

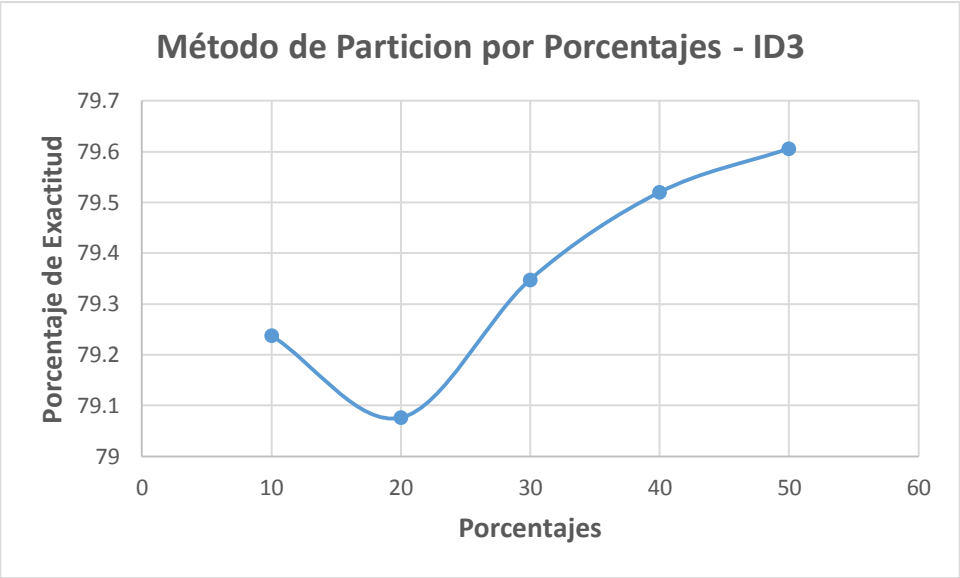



**Figura 4.14 Método Partición por Porcentaje - Exactitud – KNN**

Se puede observar que el porcentaje de exactitud oscila entre el 79.07% y el 79.60% con lo cual aunque la gráfica se ve que sube conforme se le aumenta el porcentaje al conjunto prueba en realidad no hay un gran cambio.

Si comparamos el resultado obtenido con el método tradicional y el método de Partición por Porcentajes el método tradicional arroja mejores resultados en el porcentaje de exactitud como se muestra en la **Figura 4.15**

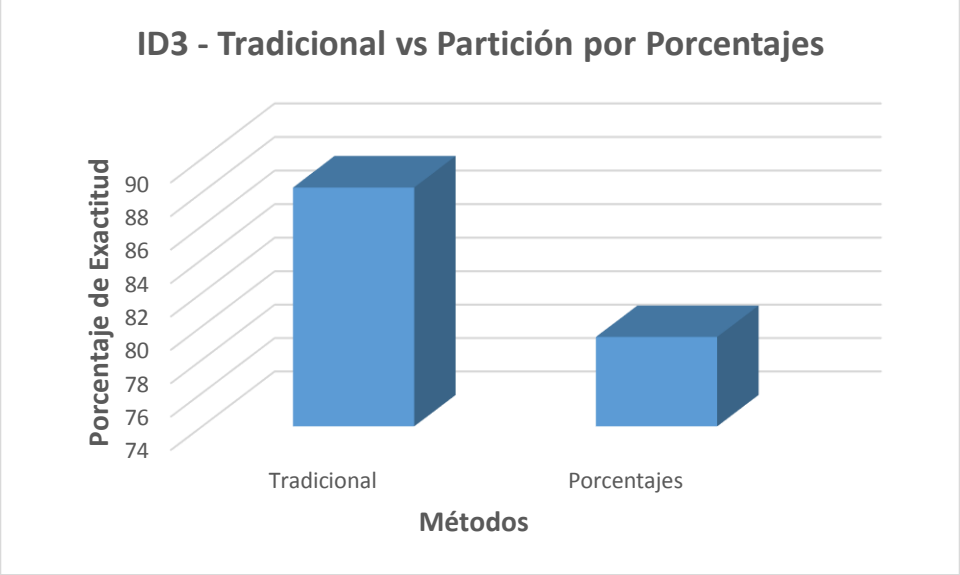



**Figura 4.15 Partición por Porcentaje vs Tradicional – ID3 - exactitud**

Como se observa se obtuvo un porcentaje de exactitud del 88.28% con el método tradicional y un 79.25% con el método de Partición por Porcentajes, siendo mejor el método tradicional con el clasificador ID3.

Ahora se realizara una comparación de los tres métodos utilizados con cada uno de los clasificadores para ver cuál fue el mejor de ellos.

**Comparando los 3 métodos utilizando Bayes Ingenuo**

Comparamos el método tradicional, el método de validación cruzada y el método de Partición por porcentajes utilizados con el clasificador de Bayes Ingenuo como se muestra en la **Figura 4.16**

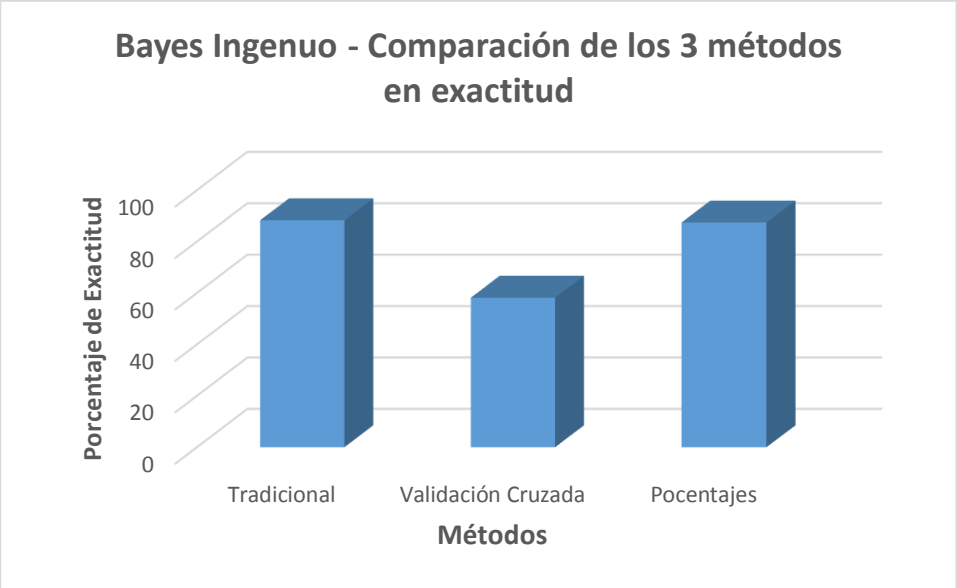



**Figura 4.16 Comparando los 3 métodos- BI - exactitud**

Como se observa en la gráfica de barras vemos que el que obtuvo mayor porcentaje de exactitud fue el método tradicional con un 88.28% aunque el método de Partición por Porcentajes también obtuvo muy buenos resultados con un 87.40% con lo cual es mínima la diferencia entre estos dos métodos.

**Comparando los 3 métodos utilizando K vecinos más cercanos**

Comparamos el método tradicional, el método de validación cruzada y el método de Partición por porcentajes utilizados con el clasificador de K vecinos más como se muestra en la **Figura 4.17**

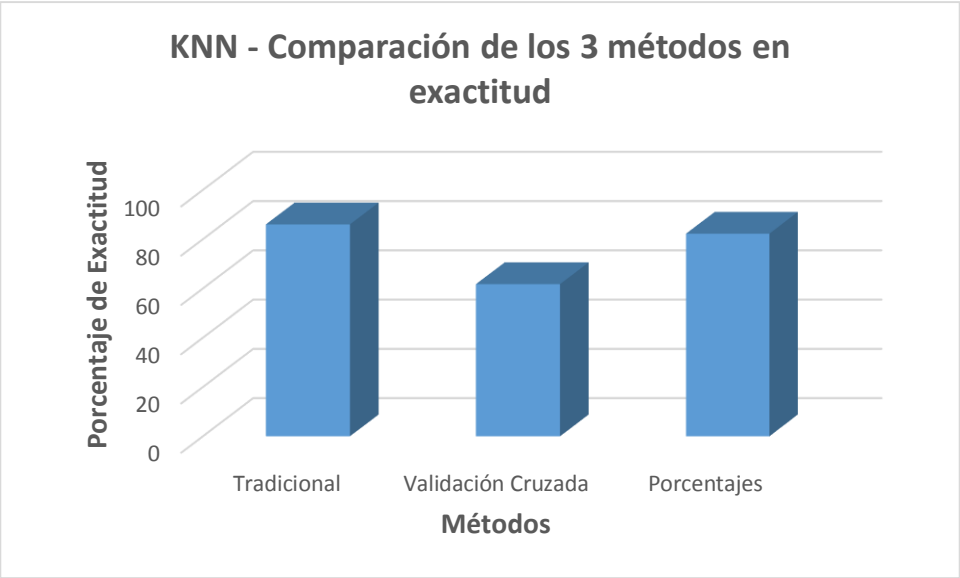



**Figura 4.17 Comparando los 3 métodos- KNN - exactitud**

Como se observa en la gráfica de barras vemos que el que obtuvo mayor porcentaje de exactitud fue el método tradicional con un 86.00% y el segundo mejor fue el método de Partición por Porcentajes obteniendo un 82.23% y por último el método de Validación Cruzada con un 61.75%.

**Comparando los 3 métodos utilizando ID3**

Comparamos el método tradicional, el método de validación cruzada y el método de Partición por porcentajes utilizados con el clasificador de K vecinos más como se muestra en la **Figura 4.18**

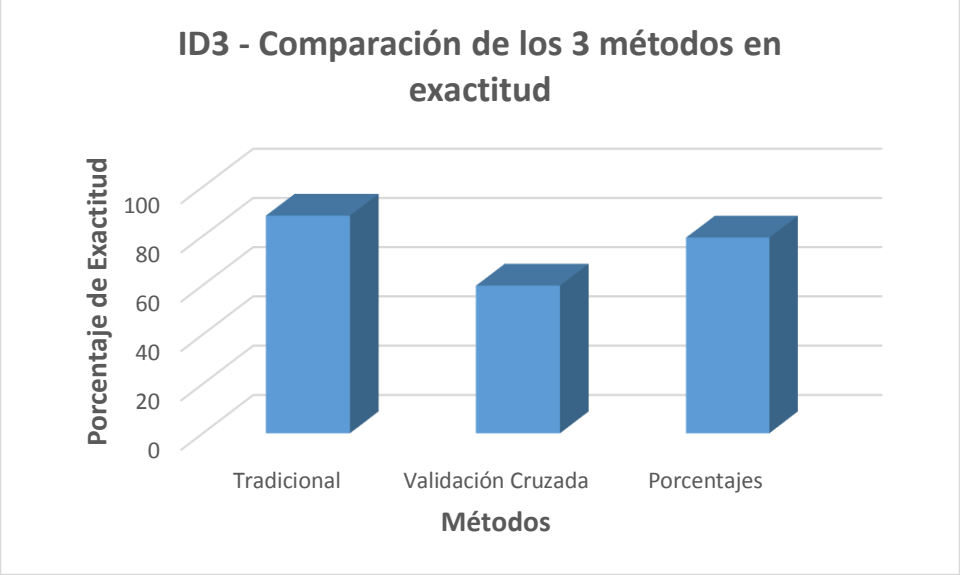



**Figura 4.18 Comparando los 3 métodos- ID3 - exactitud**

Como se observa el porcentaje de exactitud del método tradicional es el mejor con un 88.28%, mientras que el método de Partición por porcentajes tiene un 79.35% y Validación Cruzada un 59.82%.

Vamos a tomar cada método que se comportó mejor con cada uno de los clasificadores para comparar cual fue el mejor de los tres clasificadores utilizando el método de clasificación que le dio mejores resultados con lo cual podemos observar esta comparación en la **Figura 4.19**.

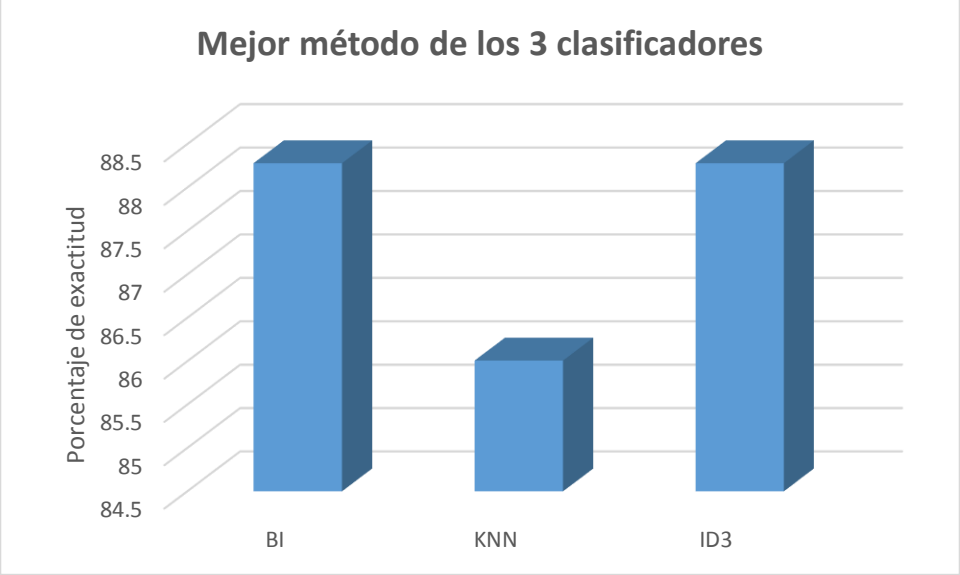



**Figura 4.19 Comparando el mejor método de cada clasificador**

Como se observó a lo largo de estas comparaciones con cada uno de los clasificadores siempre el método con el cual se obtuvieron mejores porcentajes de exactitud fue el método tradicional. Pero el que obtuvo mayor porcentaje de exactitud de os tres clasificadores fueron Bayes Ingenuo e ID3, ya que en ambos se obtuvo un 88.28% en el porcentaje de exactitud.

#### 4.2.1.4. Método Tradicional analizando el tiempo

Como ya se mencionó este método consiste en tomar el 100% de los elementos para el conjunto entrenamiento y el 100% de elementos para el conjunto prueba.

**Resultado de experimentos con Bayes Ingenuo**

Después de realizar una serie de experimentos el resultado obtenido con el clasificador Bayes Ingenuo utilizando el método tradicional se muestra en la **Tabla 4.5**

| Bayes Ingenuo con Método Tradicional | |
|---|---|
| Tiempo | 119030.934 milisegundos |

**Tabla 4.5 Método Tradicional con BI – Tiempo**

**Resultado de experimentos con K-vecinos más cercanos**

El resultado obtenido con el clasificador K-vecino más cercano al utilizar el método tradicional, se muestra en la **Tabla 4.6**

| K- vecinos más cercanos con Método Tradicional | |
|---|---|
| Exactitud | 1834105406 milisegundos |

**Tabla 4.6 Método Tradicional con KNN -tiempo**

Como se observa en la Tabla 4.3 el clasificador k-vecinos más cercano obtuvo el 86.008% de exactitud, utilizando el método tradicional, lo cual es mucho mayor a lo obtenido con Bayes Ingenuo y esto se debe a que al algoritmo de K vecinos más cercanos realiza muchas más operaciones para la clasificación.

**Resultado de experimentos con ID3**

El resultado obtenido con el clasificador ID3 al utilizar el método tradicional se muestra en la **Tabla 4.7**

| ID3 con Método Tradicional | |
|---|---|
| Exactitud | 19018.02 milisegundos |

**Tabla 4.7 Método Tradicional con ID3 - Tiempo**

Observando las Tablas 4.5, 4.6 y 4.7, podemos observar que el clasificador que obtuvo menor tiempos en clasificar es ID3.

**4.2.1.5. Método validación Cruzada analizando el tiempo**

**Resultado de experimentos con Bayes Ingenuo**

El resultado obtenido con el clasificador Bayes Ingenuo al utilizar el método de validación cruzada se muestra en la **Figura 4.20**

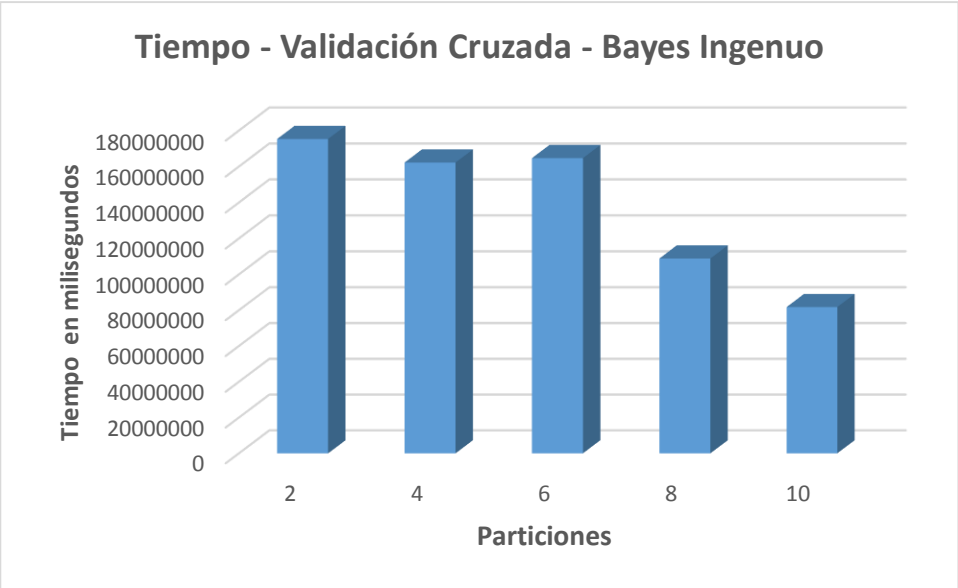



**Figura 4.20 Método Validación Cruzada - BI- Tiempo**

Como se puede observar en la gráfica de barras el tiempo es mayor con el número de particiones pequeñas, el tiempo disminuye si el número de particiones es mayor. Si comparamos el promedio del tiempo que se obtuvo al utilizar el método de Validación Cruzada con el método tradicional se tiene lo que se muestra en la **Figura 4.21**

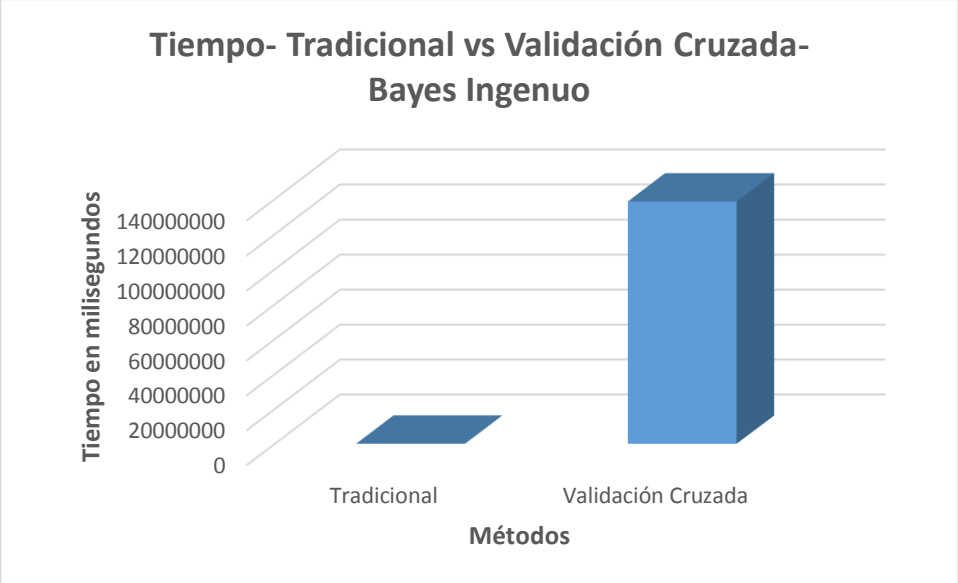



**Figura 4.21 Método Tradicional vs Validación Cruzada - BI- Tiempo**

Como se observa es muy grande la diferencia el tiempo entre el método tradicional y el método de validación cruzada, ya que con el método tradicional la clasificación tardo un total de 119030.934 milisegundos, mientras que el método de validación cruzada tardo 138577480.8 milisegundos.

**Resultado de experimentos con K-vecinos más cercanos**

El resultado obtenido con el clasificador K vecinos más cercanos al utilizar el método de validación cruzada se muestra en la **Figura 4.22**

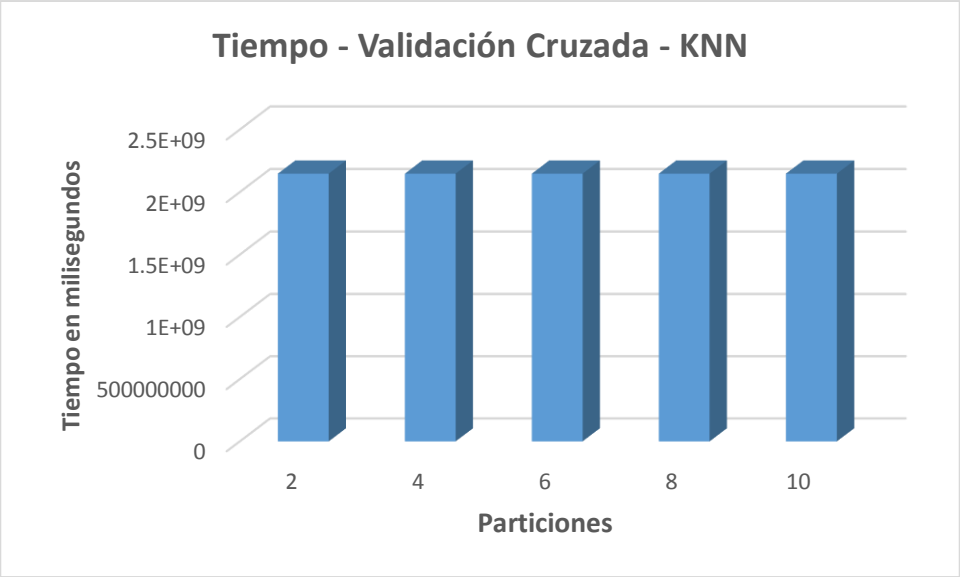



**Figura 4.22 Método Validación Cruzada - KNN- Tiempo**

Como se observa en la figura todos los experimentos que se realizaron con el método de validación Cruzada en K vecinos más cercanos se obtuvo el mismo tiempo en todas las pruebas realizadas que es de 2147483647 milisegundos, aunque el número de vecinos o el número de particiones era distinto siempre arrojaba el mismo tiempo.

Si comparamos el promedio del tiempo que se obtuvo con el método de Validación Cruzada con el método Tradicional obtenemos lo que se muestra en la **Figura 4.23**

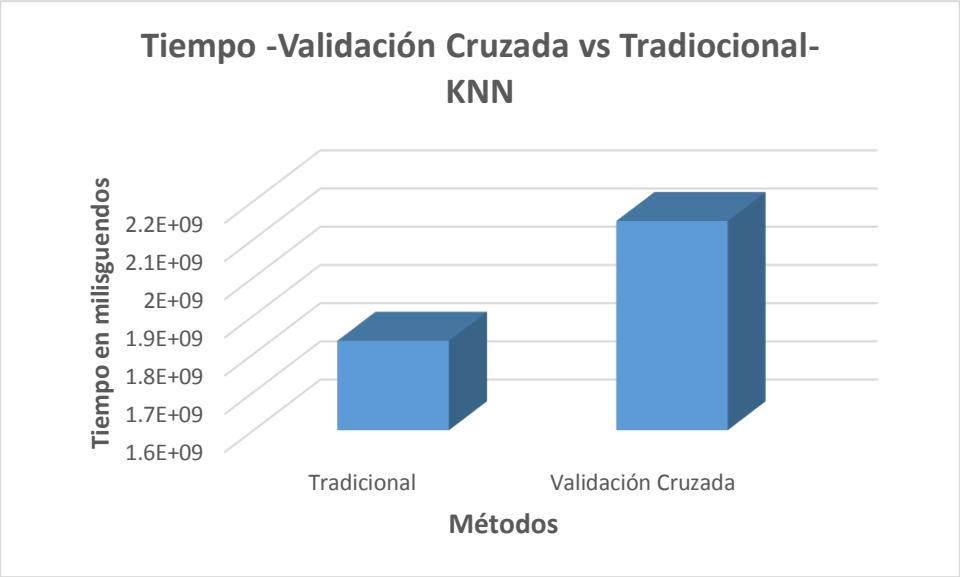



**Figura 4.23 Método Tradicional vs Validación Cruzada - KNN- Tiempo**

Como se observa en la gráfica de barras el tiempo de clasificación es menor cuando se utiliza el método tradicional con K vecinos más cercanos que con el método de Validación Cruzada.

**Resultado de experimentos con ID3**

El resultado obtenido con el clasificador ID3 al utilizar el método de validación cruzada se muestra en la **Figura 4.24**

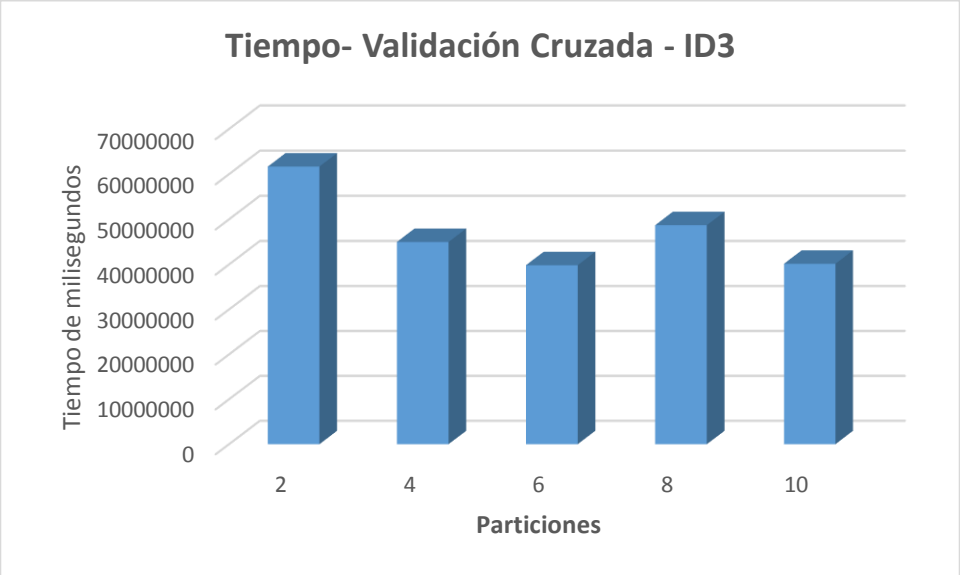



**Figura 4.24 Método Validación Cruzada – ID3- Tiempo**

Como se observa en la gráfica de barras obtenida el tiempo es menor cuando la partición es mayor utilizando el clasificador ID3.Si comparamos el promedio del tiempo que se obtuvo con el método de Validación Cruzada con el método Tradicional utilizando el clasificador ID3 obtenemos lo que se muestra en la **Figura 4.25**

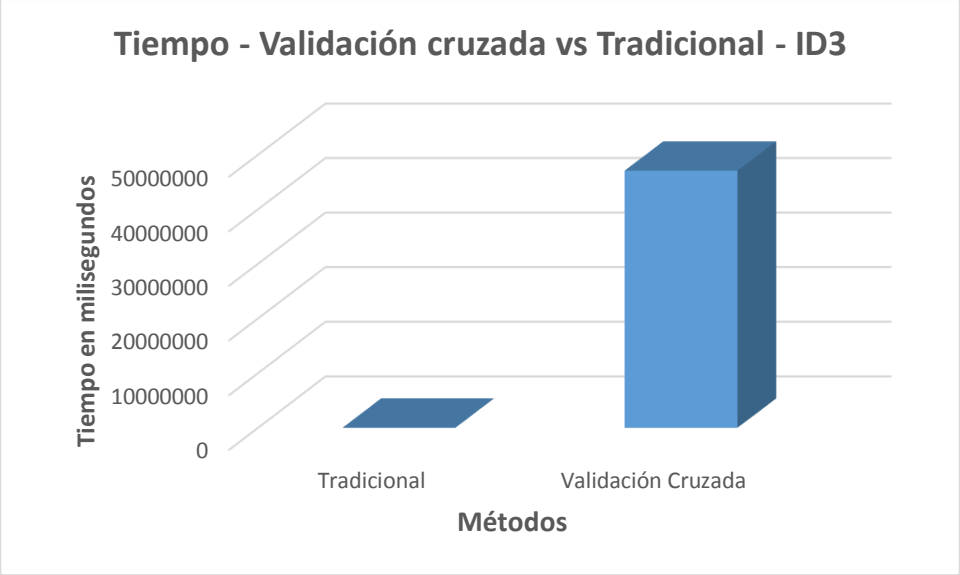



**Figura 4.25 Método Tradicional vs Validación Cruzada – ID3 – Tiempo**

Como se observa es muy grande la diferencia el tiempo entre el método tradicional y el método de validación cruzada ya que con el método tradicional la clasificación tardo un total de 19018.02 milisegundos, mientras que el método de validación cruzada tardo 47069773.92 milisegundos.

#### 4.2.1.6. Método Partición por Porcentaje analizando el tiempo

**Resultado de experimentos con Bayes Ingenuo**

El resultado obtenido con el clasificador Bayes Ingenuo al utilizar el método de Partición por Porcentaje se muestra en la **Figura 4.26**

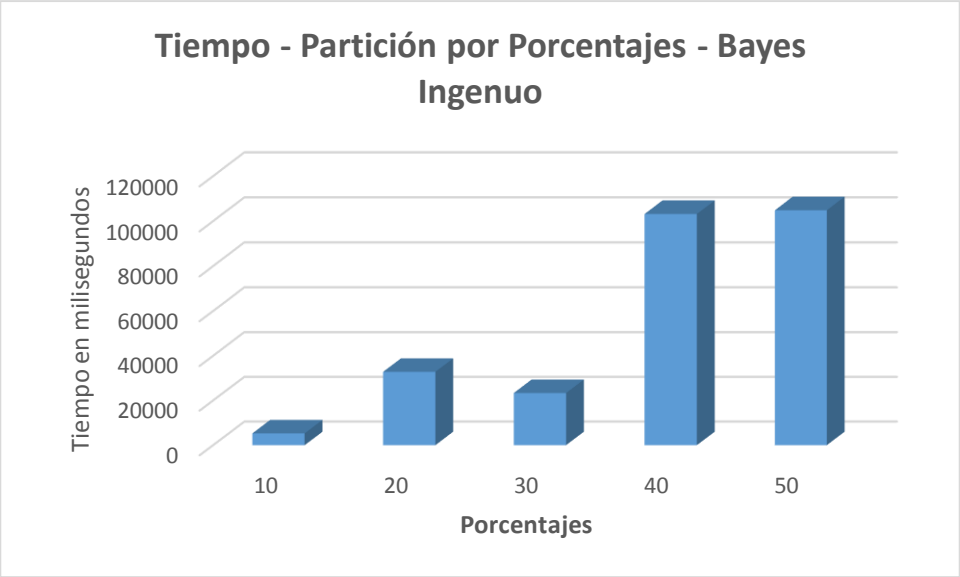



**Figura 4.26 Método Partición por Porcentajes – BI- Tiempo**

El tiempo en clasificar fue menor cuando la partición de porcentaje es menor, y el tiempo aumenta si el porcentaje aumenta, ahora vamos a comparar los tres métodos de clasificación utilizando Bayes Ingenuo para ver cuál es el mejor en tiempo. Esta comparación se muestra en la **Figura 4.27**

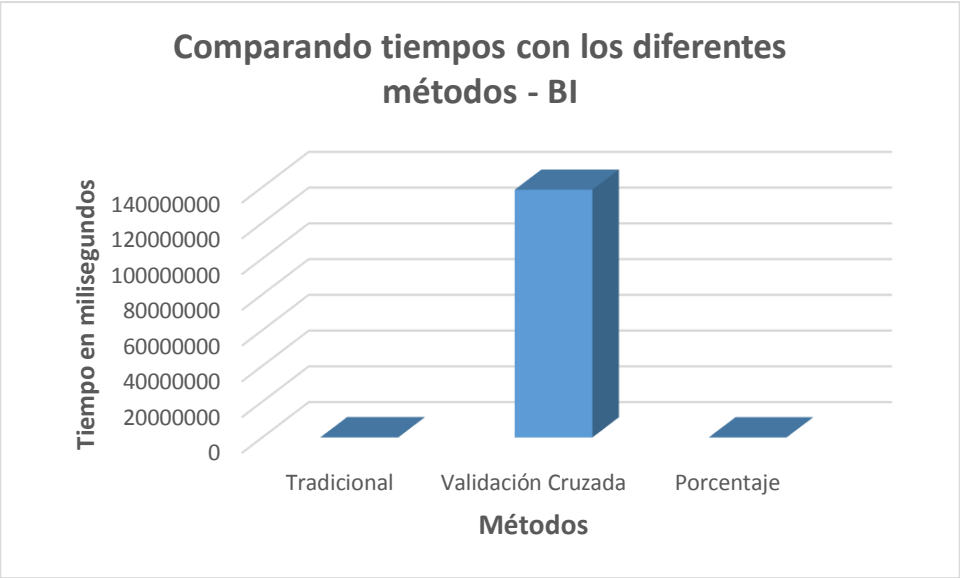



**Figura 4.27 Comparando los 3 métodos – BI – Tiempo**

Como se observa la clasificación que más se tardó utilizando Bayes Ingenuo fue con Validación Cruzada. En esta gráfica no se aprecia la diferencia entre el tiempo entre el método tradicional y el de Partición por Porcentajes ya que el tiempo del método de Validación Cruzada es mucho mayor que estos. Por lo cual se realiza la comparación entre el método tradicional y el de Partición por Porcentaje que se muestra en la **Figura 4.28**

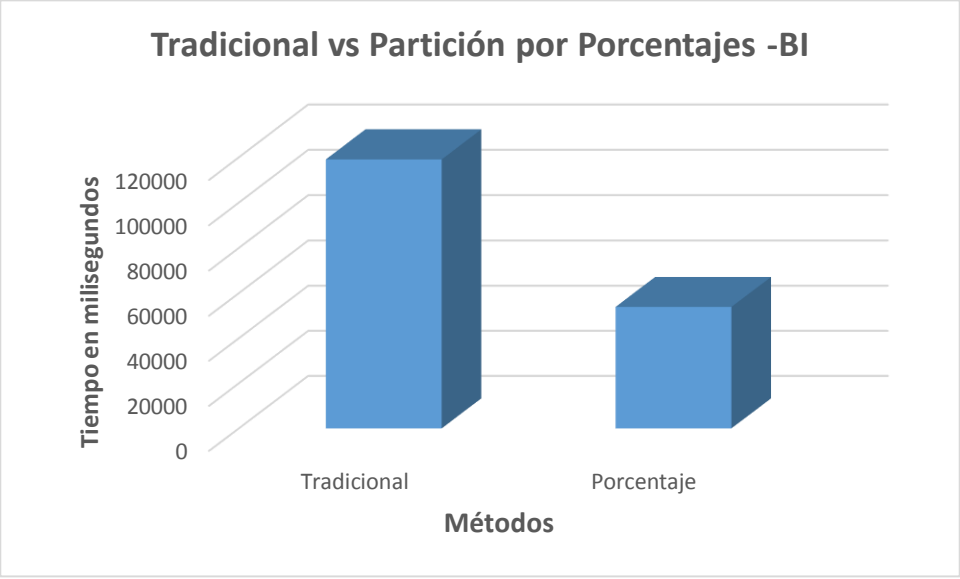



**Figura 4.28 Tradicional vs Partición por Porcentajes – BI – Tiempo**

El tiempo que tardo con el método tradicional fue de 119030.934 milisegundos, mientras que con el método de Partición por porcentaje fue de 53882.33036, concluyendo que el método que tarda menos tiempo utilizando Bayes Ingenuo es el de Partición por Porcentaje.

**Resultado de experimentos con K-vecinos más cercanos**

El resultado obtenido con el clasificador K vecinos más cercanos al utilizar el método de Partición por Porcentaje se muestra en la **Figura 4.29**

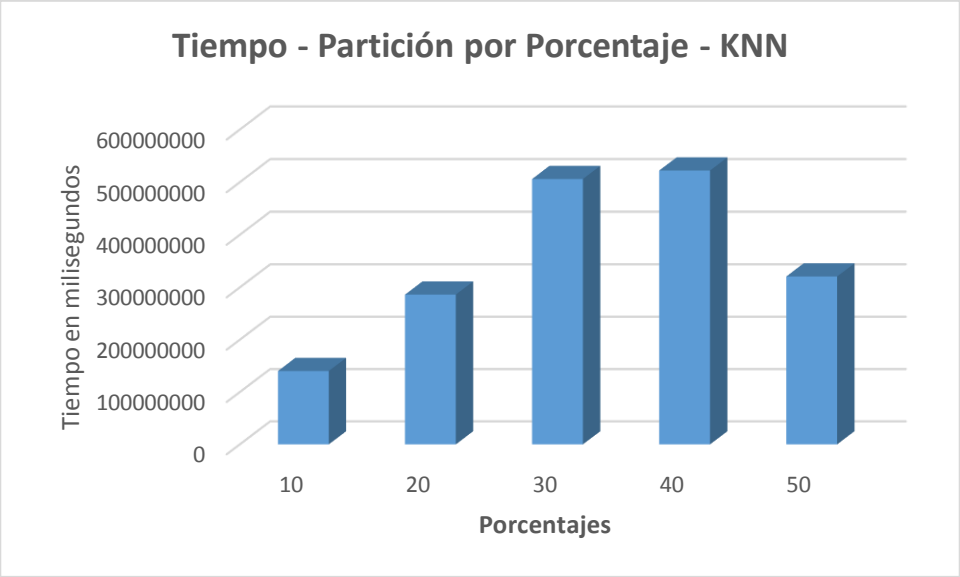



**Figura 4.29 Método Partición por Porcentajes – KNN- Tiempo**

Observando que el tiempo es menor cuando el tamaño del porcentaje es menor y este empieza a crecer conforme el porcentaje aumenta pero luego vuelve a de crecer. Ahora se sacara el promedio de estos resultados para compararlo con los tiempos que se obtuvieron en el método tradicional y el de validación Cruzada como se muestra en la **Figura 4.30**

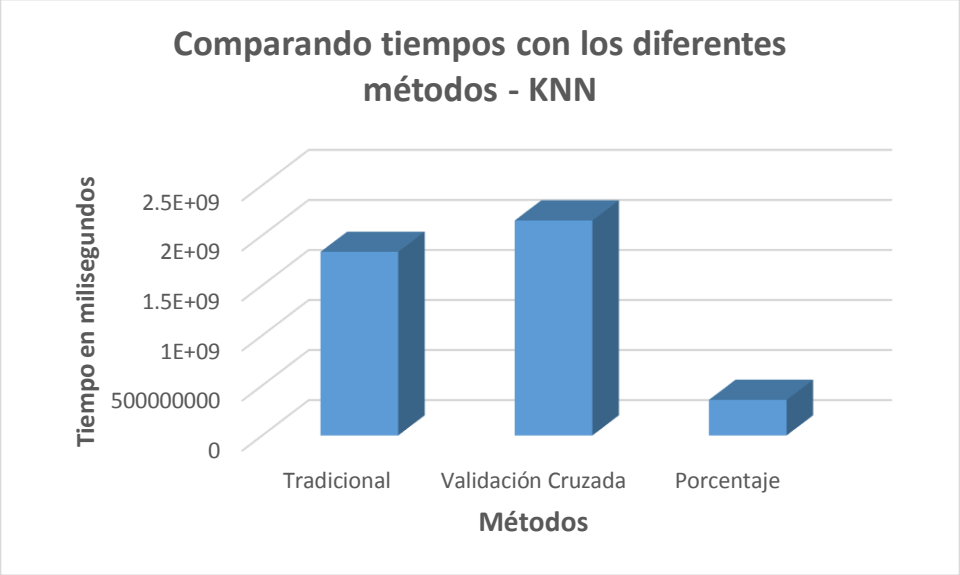



**Figura 4.30 Comparando los 3 métodos – KNN – Tiempo**

Concluyendo que el tiempo menor que se tardó K vecinos más cercanos en clasificar es con el método de Partición por Porcentaje ya que este tardó 355066414.4 milisegundos, mientras que el método tradicional tardó 1834105406 milisegundos y el método de Validación Cruzada tardo 2147483647 milisegundos.

**Resultado de experimentos con ID3**

El resultado obtenido con el clasificador ID3 al utilizar el método de Partición por Porcentajes se muestra en la **Figura 4.31**

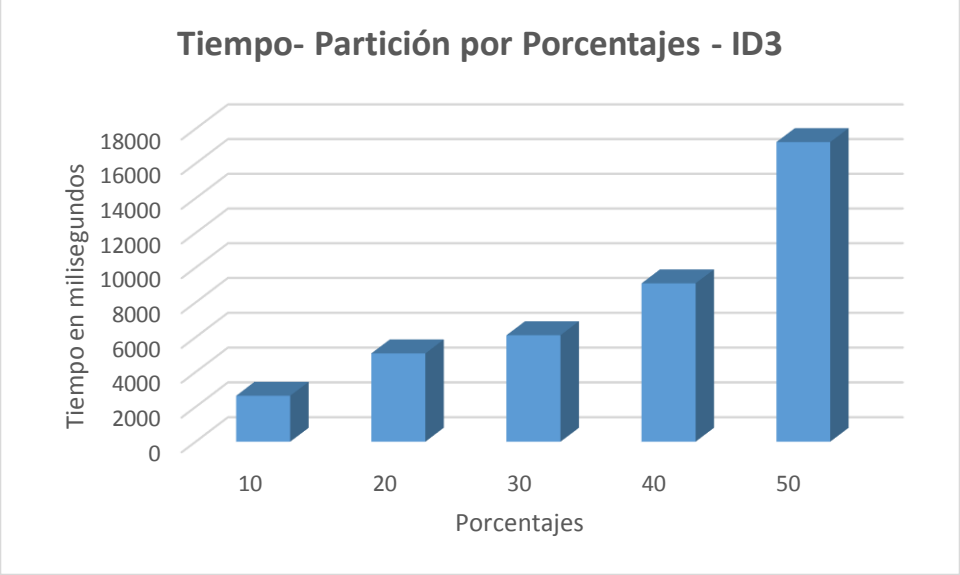



**Figura 4.31 Método Partición por Porcentajes – ID3 - Tiempo**

En la cual se observa que a mayor porcentaje que se le da el tiempo va aumentando de forma exponencial y esto se debe a que a menor número de porcentaje para el conjunto prueba mayor número de elementos para el conjunto entrenamiento, con lo cual tiene que hacer más operaciones para construir el árbol.

Al comparar el promedio de tiempo que se obtuvo con el método de Partición por Porcentajes con los otros 2 métodos se obtiene lo que se muestra en la **Figura 4.32**

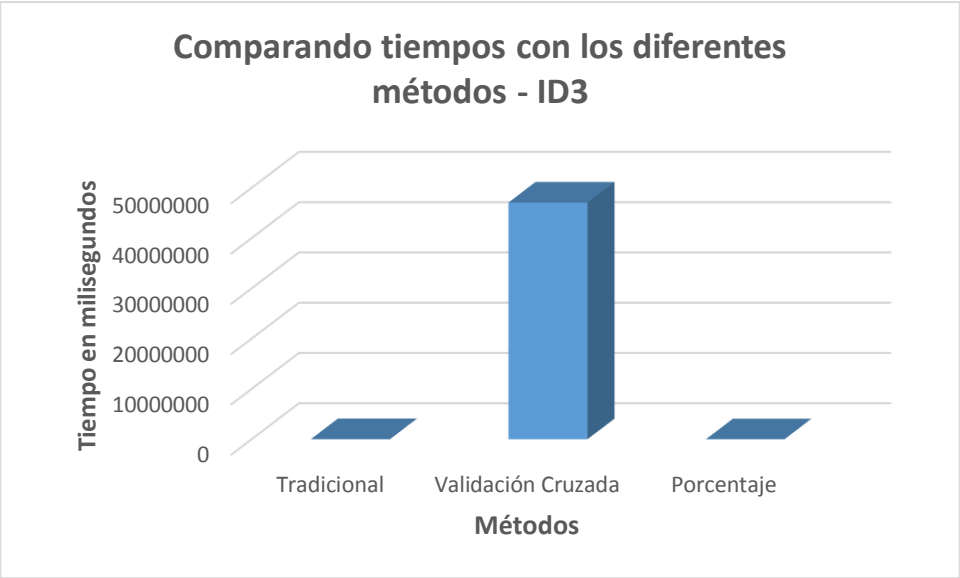



**Figura 4.32 Comparando los 3 métodos – ID3 – Tiempo**

Sobresaliendo el Método de validación Cruzada utilizando ID3 como el método que más toma tiempo para clasificar. Ya que no se puede apreciar la diferencia entre los tiempos del método tradicional y el de Partición por Porcentaje ya que son mucho menores que el de Validación Cruzada se comparan estos dos para poder concluir cual es el que requiere menor tiempo como se muestra en la **Figura 4.33**

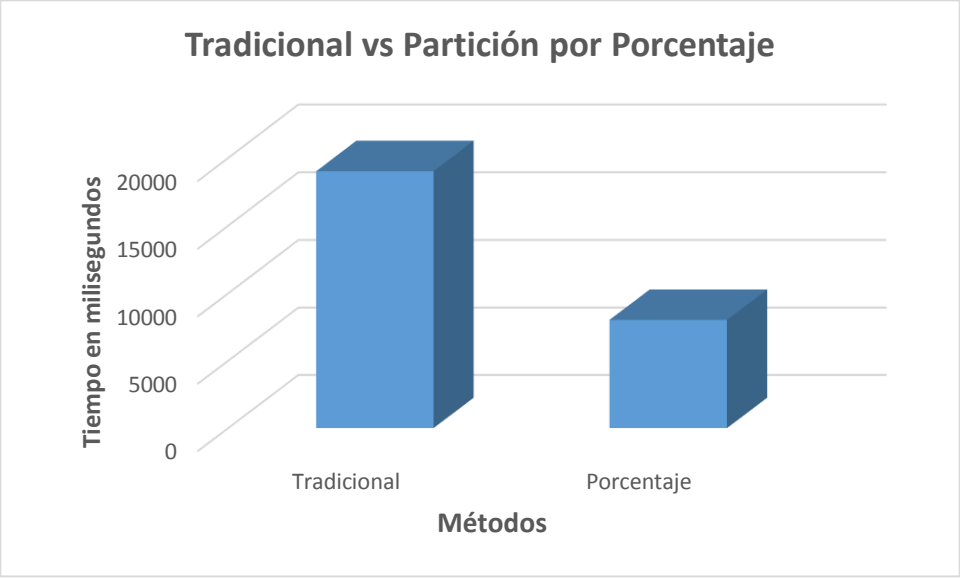



**Figura 4.33 Tradicional vs Partición por Porcentaje – ID3 – Tiempo**

Concluyendo que el método con el cual se obtuvo el menor tiempo en clasificar utilizando ID3 fue el método de Partición por Porcentaje ya que este obtuvo un tiempo de 8031.23604 milisegundos mientras que el método tradicional obtuvo un tiempo de 19018.02 milisegundos.

**Comparando los tiempos de los mejores métodos de cada clasificador**

Se realizara una comparación de los mejores tiempos obtenidos con cada uno de los clasificadores, es decir tenemos que seleccionar el método en el cual se obtuvo el menor tiempo con cada clasificador para poder concluir cual fue el clasificador que obtuvo menor tiempo en clasificar utilizando el método en el que obtuvo el menor tiempo.

Con cada uno de los clasificadores el método con el cual se obtuvo el menor tiempo fue el método de Partición por Porcentaje. Se realiza esta comparación en la **Figura 4.34**

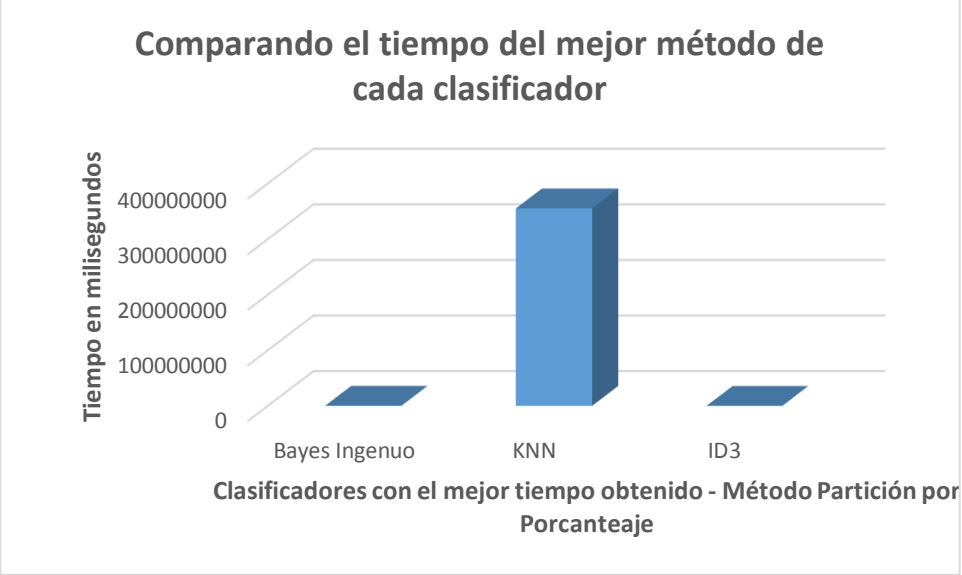



**Figura 4.34 Comparando el tiempo del mejor método de cada clasificador- Tiempo**

Como se observa el método que tardo más tiempo es el de Partición por Porcentaje utilizando K vecinos más cercanos, siendo este mucho más alto que los mejores métodos de Bayes Ingenuo e ID3, pero ya que es tanta la diferencia que se tiene con estos no se puede apreciar cual es mejor entre Bayes Ingenuo e ID3, con lo cual se realiza la comparación de estos en la **Figura 4.35**

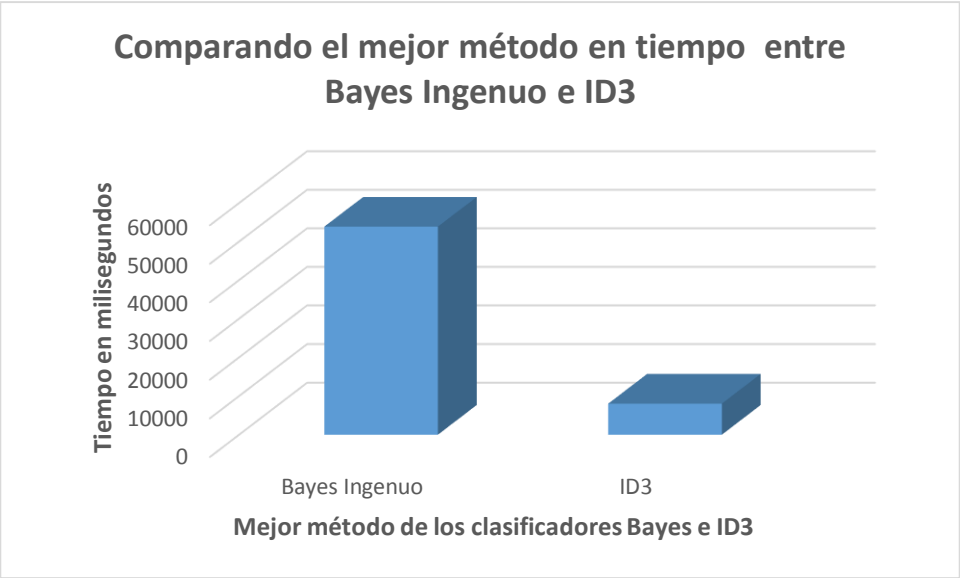



**Figura 4.35 Comparando el tiempo del mejor método de Bayes e ID3- Tiempo**

Con lo cual concluimos que el menor tiempo en clasificar se obtiene con el método de Partición por Porcentaje utilizando el clasificador ID3, ya que este obtuvo un tiempo de 8031.23604 milisegundos mientras que con Bayes Ingenuo se obtuvo un tiempo de 53882.33036 milisegundos.

# Conclusiones y Trabajo a futuro

## 5.1. Objetivos

Este trabajo tuvo como objetivo llevar a cabo el proceso de Minería de Datos para el análisis de datos obtenidos en un Reactor de Microactividad de Cracking Catalítico. Se revisaron los principales aspectos teóricos necesarios para realizar el análisis de estos datos, aplicando las fases de extracción de conocimiento sobre dicho repositorio ya que los datos se encontraban en un estado crudo, para esto fue necesario la implementación de varios clasificadores que nos permitieran llevar a cabo el proceso de clasificación sobre los la Base de Datos obtenida del proceso KDD y se calculó la medida de funcionamiento exactitud para evaluar que tan bueno eran los modelos de clasificación.

## 5.2. Desarrollo

El principal problema que tuvimos fue obtener una Base de Datos con el formato que necesitábamos para poder llevar a cabo la clasificación con los diferentes modelos de clasificación ya que los datos eran de un Reactor de Microactividad de Cracking Catalítico y se encontraban en un estado crudo, por lo cual se llevó a cabo el proceso KDD aplicando las fases de integración, limpieza, selección y transformación con lo cual se obtuvo la Base de Datos con el formato necesario.

Una vez que obtuvimos el formato adecuado de la Base de Datos realizamos una serie de pruebas con los modelos de clasificación, Bayes Ingenuo, K vecinos más cercanos e ID3 con los diferentes métodos de clasificación: Tradicional, Validación Cruzada y Partición por Porcentaje evaluando la exactitud obtenida con cada uno de ellos así como el tiempo de clasificación.

## 5.3. Logros

Ahora entendemos cómo es que se lleva a cabo el proceso KDD cuando se tienen datos crudos los cuales nos aportan ningún beneficio, obteniendo al finalizar este proceso una Base de Datos con la cual al llevar a cabo la Minería de estos datos obtenemos patrones los cuales nos permiten obtener conocimiento potencialmente útil.

## 5.4. Trabajo Futuro

Una de las cosas que quedaron inconclusas por falta de entendimiento al tema fue el modelo de clasificación C4.5, el cual estaba contemplado realizarlo a un inicio de este trabajo, con lo cual sería muy interesante implementarlo y probarlo con la Base de datos obtenida después del proceso KDD.