*B*ENEMÉRITA
*U*NIVERSIDAD *A*UTÓNOMA DE *P*UEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

DETECCIÓN DE INTRUSOS EN UNA RED DE COMPUTADORAS

TESIS PRESENTADA COMO REQUISITO PARA
OBTENER EL TITULO DE:
INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
ARTURO NIÑO REYES

ASESOR:

Dr. IVO HUMBERTO PINEDA TORRES

JUNIO 2010

AGRADECIMIENTOS

Con la mayor gratitud por los esfuerzos realizados para que yo lograra terminar mi carrera profesional siendo para mi la mejor herencia.

A mi madre que es el ser más maravilloso de todo el mundo. Gracias por el apoyo moral, tu cariño y comprensión que desde niño me has brindado, por guiar mi camino y estar junto a mi en los momentos más difíciles.

A mi padre porque desde pequeño ha sido para mi un gran hombre maravilloso al que siempre he admirado.

A mi asesor ya que sin su guia y el tiempo que me dedico no hubiera podido concluir este trabajo, gracias por la enseñanza y la ayuda que me brindó.

A mis amigos estuvieron conmigo compartiendo experiencias, desveladas, peleas y alegrías, por su apoyo incondicional en las buenas y en las malas, he aprendido tanto de ustedes que siempre los voy a recordar.

Y a todas aquellas personas que comparten conmigo este triunfo.

Con amor, admiración y respeto.

Índice

Capítulo 1

Introducción

La Minería de Datos es aplicada a una amplia gama de problemas, ya sea para predecir, caracterizar o clasificar datos. Últimamente, la Minería de Datos ha sido utilizada en el campo de la seguridad informática con el propósito de detectar intrusiones en una red de computadoras. Una de las razones son los patrones que se pueden obtener al analizar una gran cantidad de datos, para lo cual la Minería resulta de gran ayuda.

Existen dos grandes ramas para dividir la detección de intrusiones dentro de una red. La primera es denominada “Misuse Detection”, que consiste en que a cada instancia se le asigna una etiqueta como normal o intrusión y el algoritmo es entrenado con esos datos. Las etiquetas son asignadas manualmente, por lo que no es posible detectar ataques novedosos sino han sido entrenados para detectarlos, sin embargo cuando ya están clasificados su calidad para detectar es muy alta. La segunda es “Anomaly Detection”, esta consiste en detectar comportamientos anómalos o cosas que se desvíen del comportamiento normal. Esta técnica es muy buena para detectar ataques que no habían sido considerados con anterioridad. Su principal desventaja es que puede generar demasiadas falsas alarmas.

El trabajo que se presenta, consistirá en detectar el comportamiento normal de la red, para posteriormente detectar comportamientos anómalos de la red. Por lo que se clasificaran los datos para detectar de acuerdo a los algoritmos de Minería de datos cuando se presenta una intrusión o un acceso normal.

1.1 Planteamiento del problema

Caracterizar el tráfico de la red, para posteriormente detectar comportamiento anómalo dentro de la red y poder determinar si es una intrusión o un acceso normal.

1.1.1 Características del Proyecto

El proyecto trata de detectar intrusiones dentro de una red de computadoras a través del monitoreo de la red y encontrando patrones que caractericen su comportamiento utilizando Minería de Datos. En este caso se utilizaran algoritmos de clasificación para encontrar patrones interesantes entre los datos utilizados. La decisión de utilizar estos algoritmos fue porque la información es apta para el tipo de análisis que se realiza con los datos estructurados y porque se pueden establecer relaciones entre los atributos.

1.2 Objetivos generales

Este trabajo tiene por objeto aplicar los algoritmos de clasificación de minería de datos para poder determinar el ataque o acceso normal a una red de computadoras, estos algoritmos nos ayudaran a determinar de manera mas precisa el tipo de acceso de las conexiones para clasificarlas.

1.3 Objetivos específicos

El objetivo de este trabajo es el estudio y propuesta de un método de selección de atributos que se pueda aplicar a bases de datos de elevada dimensión en el marco del aprendizaje supervisado, en concreto para la tarea de clasificación. La selección de atributos se puede considerar como un problema de búsqueda en un cierto espacio de estados, donde cada estado corresponde con un subconjunto de atributos, y el espacio engloba todos los posibles subconjuntos que se pueden generar. El proceso de selección de atributos puede entenderse como el recorrido de dicho espacio hasta encontrar un estado (combinación de atributos) que optimice alguna función definida sobre un conjunto de atributos. En general, un algoritmo de selección consta de dos componentes básicos: medida de evaluación y método de búsqueda.

En este trabajo se realizaran pruebas con los datos, para extraer los resultados que se analizaran para visualizar la forma en que se llegara a la solución de encontrar un algoritmo que se adapte mas a la detección de intrusos en una red de computadoras, aplicando diferentes técnicas de Minería de Datos tales como:

- Árboles de decisión J48 y Random Forest
- Naive Bayes
- ZeroR
- Ridor

Capítulo 2

Sistemas de Detección de Intrusos

La seguridad en los computadores ha sido estudiada como una disciplina desde 1970. Y se refiere a las medidas y controles que protegen los sistemas de información contra la negación de servicio y la ausencia de autorización para revelar, modificar, o destruir los sistemas de información y datos. Sin embargo, podemos entender como seguridad una característica de cualquier sistema, informático o no, que nos indica que ese sistema está libre de todo peligro, daño o riesgo, y que es, en cierta manera, infalible. Como esta característica, particularizando para el caso de sistemas operativos o redes de computadores, es muy difícil de conseguir, según la mayoría de estudios e investigaciones es imposible; se suaviza la definición de seguridad y se pasa a hablar de fiabilidad, como probabilidad de que un sistema se comporte tal y como se espera de él, más que de seguridad; por tanto, se habla de sistemas fiables en lugar de hacerlo de sistemas seguros.

La detección de intrusos es el área aplicada de la seguridad informática encargada de informar eventos que puedan tener lugar en un sistema informático y pueda ser considerado, por unas u otras razones, como parte de un intento de intrusión. Como intrusión se entiende la realización de un acto no autorizado, como pueda ser el acceso a un sistema, la ejecución de programas no autorizados o el ataque a una red informática.

El concepto de intrusión está cercano al de un ataque dirigido, pero algunas de las tareas previas a un ataque, como pueda ser la recopilación de información o la búsqueda de servicios no está directamente tipificada como ataque.

Actualmente existe una controversia sobre si dichas actividades constituyen o no una actividad ilegal como lo pueda ser el acceso sin autorización a un sistema informático.

El hecho de que estemos ante un entorno complejo, formado por un sistema de información global interconectado a través de redes públicas de comunicación, hace difícil determinar si las actividades que realizan los sistemas por sí mismos pueden considerarse ataques cuando son realizados por personas con intención desconocida.

Un sistema de detección de intrusos ha de distinguir entre un acceso normal y habitual al sistema, que puede surgir de la puesta en marcha de servicios ofrecidos al exterior (entendiendo como exterior cualquier otro sistema ajeno al que ofrece los servicios), de un intento de vulnerar de algún modo dichos servicios, e incluso de aquellos que no debieran ser públicos, como parte del ataque a dicho sistema.

Es, por tanto, un sistema de detección de intrusos aquél capaz de advertir al administrador de todas aquellas situaciones que puedan ser consideradas como elementos o fases de una intrusión. El objetivo de dicho sistema es, en la medida de lo posible, proporcionar conocimiento de la puesta en marcha de un ataque sobre el sistema antes de que dicho ataque tenga éxito.

Se ha de considerar, por tanto, como un mecanismo previo de alarma que está indisolublemente unido al mecanismo de respuesta. De esta forma se podrán poner en marcha las medidas necesarias para mitigar el impacto.

Los sistemas detección de intrusos quedan divididos en función, fundamentalmente, del lugar donde realizan la detección. Este lugar puede ser la red, basándose en el análisis del tráfico que pasa por ésta y su contenido, o puede ser el propio sistema operativo sobre el que se monitoriza el uso que las aplicaciones, procesos y usuarios hacen de él.

Aún con un desarrollo correcto en el tratamiento de los protocolos, los detectores de intrusos de red son muy susceptibles a ataques para inutilizarlos mediante la generación de tráfico falseado que consuma su capacidad

La técnica tradicionalmente aplicada a la detección de intrusos, en todos sus ámbitos, consisten generalmente en el uso de reconocimientos de patrones para determinar ataques conocidos. De igual forma que la tecnología aplicada a la detección de virus, basada en la introducción de firmas más algunos heurísticos para detectar ligeras desviaciones, la detección de intrusos habitualmente busca en patrones que permiten discriminar un ataque de algo que no lo es.

2.1 Clasificación de ataques informáticos e intrusiones

Se pueden encontrar multitud de trabajos referentes a la categorización y clasificación de ataques informáticos e intrusiones.

Uno de los primeros trabajos dedicados a categorizar diferentes aspectos de la seguridad informática, se centraba en la debilidad de los sistemas informáticos y defectos de diseño en sistemas operativos, así como en vulnerabilidades funcionales y métodos de abusos informáticos. Varias de las taxonomías desarrolladas más tarde se enfocaban principalmente en dos aspectos: categorización de uso indebido de computadoras, y categorización de la gente que intentaba obtener acceso no autorizado a ordenadores.

Los ataques se clasifican en cuatro grupos principales, utilizando como criterio el *tipo de ataque:*

- **Denegación de Servicio (DoS):** Estos ataques tratan de detener el funcionamiento de una red, máquina o proceso; o si no denegar el uso de los recursos o servicios a usuarios autorizados. Hay dos tipos de ataques DoS; por un lado ataques de sistema operativo, los cuales tratan de explotar los fallos en determinados sistemas operativos y pueden evitarse aplicando los respectivos parches; y ataques de red, que explotan limitaciones inherentes de los protocolos e infraestructuras de red.
- **Indagación o exploración (probe):** Este tipo de ataques escanean las redes tratando de identificar direcciones IP válidas y recoger información acerca de ellas (servicios que ofrecen, sistemas operativos que usan). A menudo, esta información provee al atacante una lista de vulnerabilidades potenciales que podrían ser utilizadas para llevar a cabo ataques a los servicios y las máquinas escogidas. Estos ataques son los más frecuentes, y a menudo son precursores de otros ataques.

- **R2L (Remote to Local):** cuando un atacante que no dispone de cuenta alguna en una máquina, logra acceder (tanto como usuario o como root) a dicha máquina. En la mayoría de los ataques R2L, el atacante entra en el sistema informático a través de Internet.
- **U2R (User to Root):** Este tipo de ataque se da cuando un atacante que dispone de una cuenta en un sistema informático es capaz de elevar sus privilegios explotando vulnerabilidades en los mismos, un agujero en el sistema operativo o en un programa instalado en el sistema.

Estas categorías están solamente para la clasificación teórica puesto que en nuestros experimentos utilizamos las clasificaciones de los ataques apropiados, considerados en la siguiente Tabla que presenta los grupos de ataques, los tipos que los compongan y una descripción del ataque con el efecto causado.

Todos los ataques citados en la tabla 2.1 son proporcionados desde los archivos de registro y la base de datos.

Grupo	Nombre del ataque	Servicio	Mecanismo Utilizado	Efecto del Ataque
DOS	back	http	abuso/ vulnerabilidad	Disminuye el tiempo de respuesta del servidor
DOS	land	http	vulnerabilidad	Detiene la maquina
DOS	neptune	icmp	Abuso	Disminuye el tiempo de respuesta de la maquina
DOS	pod	icmp	vulnerabilidad	Detiene la maquina
DOS	smurf	icmp	abuso	Disminuye el tiempo de respuesta de la red
DOS	teardrop	IP	vulnerabilidad	Detiene la maquina
PROBE	ipsweep	icmp	abuso	Identifica maquinas activas
PROBE	nmap	others	abuso	Revisa puertos abiertos de una maquina
PROBE	portsweep	others	abuso	Revisa puertos abiertos de una maquina
PROBE	satán	others	abuso	Busca vulnerabilidades conocidas
R2L	ftp write	ftp	Error de configuración	Toma acceso de la maquina
R2L	guess_passwd	telnet, rlogin, pop, ftp, imap	abuso	Toma acceso de la maquina
R2L	imap	imap	vulnerabilidad	Toma acceso de la maquina como root
R2L	multihop	tcp	vulnerabilidad	Toma acceso de la maquina
R2L	phf	http	vulnerabilidad	Ejecuta comandos como usuario http
R2L	spy	ftp	Error de configuración	Toma acceso de la maquina
R2L	warezclient	tcp	vulnerabilidad	Toma acceso de la maquina
R2L	warezmaster	tcp	vulnerabilidad	Toma acceso de la maquina
U2R	buffer overflow	Sesión de usuario	Desborde de pila	Toma acceso de la maquina como root
U2R	loadmodule	Sesión de usuario	Error de configuración	Toma acceso de la maquina como root

U2R	perl	Sesión de usuario	Error de configuración	Toma acceso de la maquina como root
U2R	rootik	Sesión de usuario	vulnerabilidad	Toma acceso de la maquina como root

Tabla 2.1 Tipos de Ataque

Las características básicas del protocolo TCP/IP se muestran en la tabla 2.2 y con ella se consiguen la información esencial y elemental si se analiza un paquete de la pila de los protocolos TCP/IP

Nombre	Descripción	Tipo
Duration	Duración en segundos de la conexión	Continua
Protocol_type	Tipo de protocolo usado en la conexión	Discreta
Service	Servicio utilizado de la red	Discreta
Src_bytes	Numero de bytes enviados de la fuente al destino	Continua
Dst_bytes	Numero de bytes enviados del destino a la fuente	Continua
Flag	Estado de la conexión (normal o error)	Discreta
Land	1 si la conexión es from/to el mismo host/port; 0 de otra manera	Discreta
Wrongfragment	Numero de fragmentos incorrectos	Continua
Urgent	Numero de paquetes urgentes	Continua

Tabla 2.2 Características del protocolo TCP/IP

Características sugeridas a través del conocimiento del área. Es información extraída de los paquetes con la ayuda de especialistas. El ejemplo sería el registro de tentativas de la conexión con el usuario privilegiado en la conexión resuelta o el número del acceso llevado a través de archivos. Estas se muestran en la tabla 2.3.

Nombre	Descripción	Tipo
Hot	Numero de indicadores clave	Continua
Numfailedlogins	Tentativas de la conexión sin éxito	Continua
Logged_in	1 si su login es correcto; 0 caso contrario	Discreta
Numcompromised	Numero de condiciones comprometidas	Continua
Root_shell	1 si el root Shell es obtenido; 0 caso contrario	Discreta
Su_attempted	1 si el comando su root es atendido ; 0 caso contrario	Discreta
Numroot	Numero de acceso de root	Continua
Num_file_creations	número de las operaciones de la creación del archivo	Continua
Numshells	Numero de avisos de Shell	Continua
Num_access_files	número de operaciones en archivos de control de acceso	Continua
Num_outbound_cmds	número de comandos de salida en una sesión del ftp	Continua
Is_hot_login	1 si la conexion pertenece a la lista “hot”; 0 caso contrario	Discreta
Is_guest_login	1 si la conexión login es un guest	Discreta

Tabla 2.3 Características de la conexión

Las características conseguidas a través de una ventana de 2 segundos se muestran en la tabla 2.4:

Nombre	Descripción	Tipo
Count	número de conexiones al mismo host que la conexión actual en los últimos dos segundos	Contínua
Serrorrate	% de las conexiones que tienen errores del SYN	Contínua
Rerrorrate	% de las conexiones que tienen errores de REJ	Contínua
Same_srv_rate	% de las conexiones con algún servicio	Contínua
Diff_srv_rate	% de las conexiones con diferentes servicios	Contínua
Srv_count	número de conexiones al mismo servicio que la conexión actual en los últimos dos segundos	Contínua
Srv_serror_rate	% de las conexiones que tienen errores del SYN	Contínua
Srv_rerror_rate	% de las conexiones que tienen errores de REJ	Contínua
Srv_diff_host_rate	% de conexiones a diferentes host	Contínua

Tabla 2.4. Características de conexión

2.2 Clasificación de los Sistemas de Detección de Intrusos (IDS)

La clasificación o taxonomía de los sistemas de detección de intrusos ha sido tratada en numerosos trabajos. La clasificación más común se realiza en base a tres características funcionales de los IDS:

- **Fuentes de información.** Se refiere al origen de los datos que se usan para determinar si una intrusión se ha llevado a cabo.
- **Análisis.** Se trata del método de detección utilizado. La información recogida en el paso anterior puede ser analizada mediante diferentes estrategias.
- **Respuestas.** Una vez se ha determinado si ha sucedido alguna intrusión, los IDS pueden o bien responder de forma activa ante la misma, o bien registrar la detección y no realizar acción alguna.

2.3 Fuentes de Información

Desde el inicio de los IDS, se ha trabajado con multitud de datos provenientes de diferentes fuentes para tratar de identificar la existencia de una intrusión. Estos datos se pueden diferenciar en dos grandes grupos; aquellos datos obtenidos de una máquina o host, y aquellos datos obtenidos a partir de la monitorización de una red. Dentro de cada grupo, se pueden identificar diferentes enfoques que se pueden tomar:

Host:

- Logs o registros de auditoria
- Llamadas del sistema o system calls de procesos en ejecución
- Solaris BSM (Basic Security Module)
- Métricas de uso del sistema
- Comandos del teclado

Red:

- Comunicación de datos (Ethernet, Token Ring, …)
- Nivel de red (normalmente IP)
- Nivel de Transporte/control (TCP, UDP, RTP, ICMP,...)
- Nivel de Aplicación (HTTP, DNS, Telnet, FTP, SSH, SMTP,...)
- Wireless

Capítulo 3

MINERÍA DE DATOS

La minería de datos consiste en la "explotación" de datos en bruto. Su objetivo, perseguido mediante la manipulación automática o semiautomática de los datos, es la obtención de información clave para conseguir beneficios, información más relevante y útil que los propios datos de partida.

La minería de datos se fundamenta en la intersección de diversas áreas de estudio, entre las que cabe destacar: análisis estadístico, bases de datos, inteligencia artificial y visualización gráfica.

Una buena definición de lo que es minería de datos puede ser la siguiente:

Es el empleo de algoritmos y procedimientos para sacar a la luz asociaciones, correlaciones, reglas, patrones e incluso excepciones interesantes o potencialmente útiles, desconocidos y escondidos en bases de datos (o almacenes de datos).

A veces se ha apelado al nombre de "descubrimiento de conocimiento en bases de datos" (KDD: Knowledge Discovery in Data Bases) para hacer referencia a la minería de datos; sin embargo, muchos autores prefieren referirse al proceso de minería de datos como al de la aplicación de un algoritmo para extraer patrones de datos y a descubrimiento de conocimiento como al proceso completo.

KDD implica un proceso interactivo e iterativo, involucrando la aplicación de métodos de minería de datos, para extraer o identificar lo que se considera como conocimiento de acuerdo a la especificación de ciertos parámetros usando una base de datos, , muestreo y transformaciones de la base de datos. La meta de este proceso es procesar automáticamente grandes cantidades de datos crudos, identificar los patrones más significativos y relevantes, y presentarlos como conocimiento apropiado para satisfacer las metas del usuario.

La Figura 3.1 muestra que existen diversos pasos para el proceso de descubrimiento de conocimiento:

1. Entender el dominio de aplicación, el conocimiento relevante a usar y las metas del usuario.
2. Seleccionar el conjunto de datos y enfocar la búsqueda en subconjuntos de variables o muestras de datos donde realizar el proceso de descubrimiento.
3. Filtrar (limpiar) y pre-procesar datos, diseñando una estrategia adecuada para manejar ruido, valores incompletos, secuencias de tiempo, etc.
4. Reducir datos y proyecciones para disminuir el número de variables a considerar.
5. Seleccionar la tarea de descubrimiento a realizar, por ejemplo: clasificación, agrupamiento, regresión, etc.
6. Seleccionar el o los algoritmos a utilizar.
7. Llevar a cabo el proceso de minería de datos.
8. Interpretar los resultados y posiblemente regresar a algún paso anterior. Esto puede involucrar repetir el proceso, quizás con otros datos, otros algoritmos, otras metas y otras estrategias.
9. Incorporar el conocimiento descubierto al sistema (normalmente para mejorarlo) lo cual puede incluir resolver conflictos potenciales con el conocimiento existente.

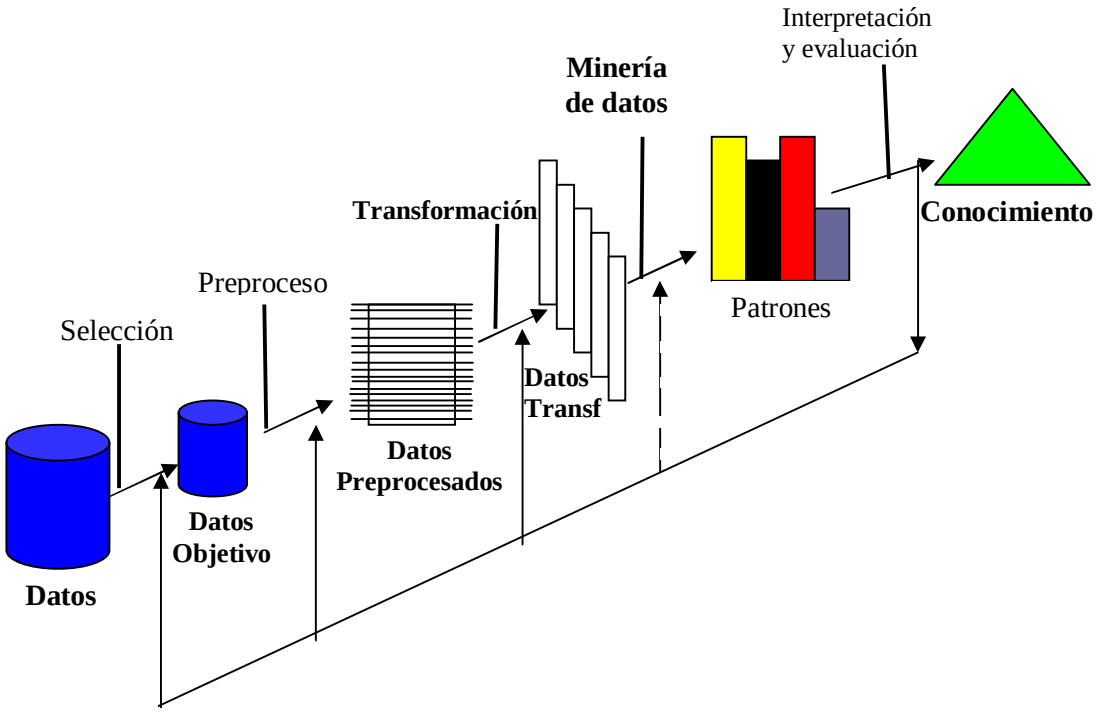


Figura 3.1 Pasos para el descubrimiento del conocimiento

Los algoritmos de minería de datos realizan en general tareas de predicción (de datos desconocidos) y de descripción (de patrones).
El término "patrón" se refiere a cualquier relación entre los elementos de la base de datos. Pueden incluir medidas de incertidumbre. Aquí se aplican una gran cantidad de algoritmos de aprendizaje y estadísticos. Un patrón es interesante en la medida que sea confiable, novedoso y útil respecto al conocimiento y los objetivos del usuario. La evaluación normalmente se le deja a los algoritmos de extracción de patrones que generalmente están basados en significación estadística. Suele decirse que un patrón representa conocimiento si su medida de interesante rebasa un cierto umbral, lo cual está basado únicamente en medidas definidas por el usuario.

Dentro de las principales tareas de la minería de datos se encuentran:

Análisis de dependencias: El valor de un elemento puede usarse para predecir el valor de otro. La dependencia puede ser probabilística, puede definir una red de dependencias o puede ser funcional. Se ha orientado mucho en los últimos años en el descubrimiento de redes bayesianas o causales en donde la dependencia se da a nivel estructural (dependencias e independencias entre variables) y cuantitativa.

Identificación de clases: ldentifica un conjunto finito de categorías o clusters que describen los datos (pueden ser exhaustivas y mutuamente exclusivas o jerárquicas y con superposiciones). Las clases pueden ser relevantes en sí o pueden servir como entradas a otros sistemas de aprendizaje. Se utilizan algoritmos de clustering. Normalmente el usuario tiene una buena capacidad de formar las clases y se han desarrollado herramientas visuales interactivas para ayudar al usuario.

Descripción de conceptos: Se resume un cierto patrón. La descripción puede ser característica (qué registros son comunes entre clases) o discriminatoria (cómo difieren las clases). La mayoría de los sistemas de aprendizaje encuentran descripciones de conceptos y están enfocados a clasificación: aprender una función que mapea (clasifica) un dato dentro de un conjunto de posibles clases predefinidas. Otra técnica relacionada es regresión: aprender una función que mapea un dato a una variable real. A veces se trata de encontrar descripciones compactas de subconjuntos de datos (media y varianza, leyes físicas) que los resuman de alguna forma.

Detección de desviaciones, casos extremos o anomalías: Detectar los cambios más significativos en los datos con respecto a valores pasados o normales. Sirve para filtrar grandes volúmenes de datos que son menos probables de ser interesantes. El problema está en determinar cuándo una desviación es significativa para ser de interés.

La minería de datos ha surgido del potencial del análisis de grandes volúmenes de información, con el fin de obtener resúmenes y conocimiento que apoye la toma de decisiones y que pueda construir una experiencia a partir de los millones de transacciones detalladas que registra una corporación en sus sistemas informáticos. La minería de datos parece ser más efectiva cuando los datos tienen elementos que pueden permitir una interpretación y explicación en concordancia con la experiencia humana.

Lo anterior se facilita mucho si estos elementos son el espacio y el tiempo. Por fortuna, se estima que el 80% de los datos registrados en una base de datos tiene la posibilidad de ubicarse espacialmente, y el 100%, de puntualizarse temporalmente. ¿Qué quiere decir esto? En primer lugar, que en la mayoría de los casos es posible asociar un punto en el espacio, un domicilio, unas coordenadas geográficas con la entidad que representa el dato, y una fecha o punto en el tiempo. En segundo lugar, que los patrones o inferencias sobre los datos son usualmente interesantes, en la medida en que son patrones en el tiempo o en el espacio. Por ejemplo, qué productos se comercializan mejor en la temporada navideña, en qué regiones es productivo sembrar tabaco, qué áreas de una zona urbana incrementarán su demanda de centros comerciales.

La tecnología promete analizar con facilidad grandes volúmenes de datos y reconocer patrones en tiempo y espacio que soportarán la toma de decisiones y construirán un conocimiento corporativo de alto nivel. La tecnología de minería de datos parece robusta y lista para su aplicación, dado el gran crecimiento de empresas que comercializan software con diferentes técnicas. Más aún, gran parte de estas técnicas son una combinación directa de madurez en tecnología de bases de datos y almacenes de datos con técnicas de aprendizaje automático y de estadística. Sin embargo, la tecnología enfrenta aún varios retos.

El primero de estos problemas, es la facilidad con que se puede caer en una falsa interpretación. Para explicarlo, basta reconocer que las primeras y más maduras técnicas para el análisis de datos, con el fin de modelar un fenómeno, provienen de la estadística. Y existe la posibilidad de ser engañados por la estadística; la información disponible para la toma de decisiones comúnmente es una serie de promedios y estimadores estadísticos que presentan una generalización de un gran volumen de datos, o una inferencia, o un estudio de correlaciones.

La estadística es una herramienta poderosa, y es elemento crucial en el análisis de datos. Sin embargo, no debe olvidarse que sus resultados se aplican a grupos o poblaciones, y no a individuos.

Esta susceptibilidad a malas interpretaciones se ve amplificada con el uso de software de minería de datos. Dichas herramientas informáticas pueden poner a disposición de un "analista" o "minero" de datos, la posibilidad de crear fácilmente indicadores, resúmenes, gráficos, y aparentes tendencias, sin un verdadero entendimiento de lo que se está reflejando. Es decir, resulta muy fácil hacer creíble una falsedad, posiblemente porque la produjo una computadora, con muchas gráficas y con base en muchos datos. Entonces, es importante que las estimaciones e inferencias sean válidas.

Tampoco debe perderse de vista que el software de minería de datos esta diseñado para hallar correlaciones, diríase que para olfatearlas. Su tarea consiste en encontrar aquella proyección de los datos, aquella perspectiva donde aparece una correlación y, lamentablemente, en muchos casos, presentarla como una relación causa-efecto. Pero el encontrar una correlación estadísticamente significativa no implica haber hallado una relación causa-efecto.

La minería de datos es una herramienta explorativa y no explicativa. Es decir, explora los datos para sugerir hipótesis. Es incorrecto aceptar dichas hipótesis como explicaciones o relaciones causa- efecto. Es necesario coleccionar nuevos datos y validar las hipótesis generadas ante los nuevos datos, y después descartar aquellas que no son confirmadas por los nuevos datos.

A esto se añaden algunos problemas. El primero, es el de minería de datos con relaciones en el tiempo. Es muy posible que se deseen hacer inferencias y análisis de datos sobre un periodo determinado, pero que durante dicho periodo no se haya registrado el mismo número de variables, o que éstas no tengan la misma precisión, o carezcan de la misma interpretación.

En ciertos casos puede que se haya hecho un ejercicio de minería de datos en el pasado y que los datos se hayan descartado o destruido, pero que se desee hacer una comparación con datos más recientes. Nótese que un ejercicio de minería de datos puede traer a la luz relevancia de variables y factores, pero que sea imposible recopilar estas variables y completar adecuadamente conjuntos de datos del pasado.

Otros problemas de análisis de datos con relación al tiempo, son asociados al carácter discreto de los datos con respecto al tiempo.

En este sentido, no se conocen todos los datos en el continuo del tiempo. Por ejemplo, si se hacen recopilaciones mensuales, es imposible hacer una predicción semanal.

Afortunadamente, existen algunas técnicas para resolver estos problemas, pero se requiere cierta madurez estadística para su comprensión. La modelación en computadora del tiempo y el espacio son problemas complejos, sobre todo para hacer inferencias. Esto hace que las técnicas de Aprendizaje Automático enfrenten mayores dificultades cuando abordan los temas que parecen más interesantes, los de descubrimiento de patrones.

La mayoría de los algoritmos de data mining se pueden ver como una combinación de unas pocas técnicas y principios. En particular, los algoritmos de data mining consisten mayormente en una mezcla específica de tres componentes:

El modelo. El componente principal. Tiene dos factores relevantes: su función (como la de clasificar, agrupar, resumir,…), y el modo de representar el conocimiento (como una función lineal de múltiples variables, en modo de árbol, de reglas, una red,…). Un modelo contiene ciertos parámetros que deben determinarse a partir de los datos.

El criterio de preferencia. Es la base para escoger un modelo o un conjunto de parámetros sobre otros. El criterio suele ser la función que hace que el modelo se ajuste más apropiadamente a los datos que se disponen.

El algoritmo de búsqueda. La especificación de un algoritmo para obtener modelos particulares y parámetros, los datos, el modelo (o familia de modelos), y un criterio de preferencia.

3.1 Funciones de Modelos

Las funciones de los modelos más comunes en el data mining incluyen:

Clasificación: clasifica un caso entre varias clases o categorías predefinidas. Los modelos de clasificación se pueden construir utilizando una gran variedad de algoritmos. Se catalogan los algoritmos de clasificación en tres tipos:

- Extensiones de discriminación lineal (como perceptrón multicapa, discriminación lógica),
- Árboles de decisión y métodos basados en reglas (como C4.5, AQ, CART), y
- Estimadores de densidad (Naïve Bayes, k-nearest neighbor, LVQ).

Regresión: clasifica un caso a una variable de predicción de valor-real. En la regresión se persigue la obtención de un modelo que permita predecir el valor numérico de alguna variable.

Clustering *(agrupamiento)*: clasifica un caso en una de las clases o agrupaciones en las que las clases se deben determinar a partir de los propios datos. Los clusters se definen buscando agrupaciones naturales de los datos basado en modelos de medidas de similaridad, densidad de probabilidad o distancia.

Summarization *(resumen)*: provee una descripción compacta de un subconjunto de datos de entrada (media y desviación estándar para todos los campos, o reglas de resumen, técnicas de visualización multivariadas, relaciones funcionales entre variables).

Modelado de dependencias: describe las dependencias significantes entre variables.

Existen modelos de dependencias a dos niveles: el estructurado y el cuantitativo. El modelo estructurado de dependencias especifica (a menudo en modo gráfico) qué variables son localmente dependientes; el modelo cuantitativo especifica la fortaleza de las dependencias usando una escala numérica. El análisis de relaciones (como reglas de asociación), que determina relaciones existentes entre elementos de una base de datos, podría considerarse un caso particular de modelado de dependencias.

Link analysis *(análisis de conexiones):* determina las relaciones o vínculos entre campos de la base de datos. El objetivo es el de deducir correlaciones entre campos satisfaciendo el umbral de confianza.

Análisis de secuencias: modela patrones secuenciales (como datos con dependencia del tiempo). El objetivo es modelar los estados del proceso generando la secuencia, o extraer y describir desviaciones y tendencias sobre el tiempo.

3.2 Representación del modelo

Modelos populares de representación incluyen reglas y árboles de decisión, modelos lineales, modelos no-lineales (como redes neuronales), métodos basados en ejemplos (como vecino más cercano y métodos de razonamiento basado en casos), modelos de dependencias gráficas de probabilidades (como redes Bayesianas), modelos de atributos relacionales.

La representación del modelo determina tanto la flexibilidad del modelo en cuanto a la representación de los datos como la interpretabilidad del modelo en términos humanos.

En la figura 3.2 se puede observar algunos de los algoritmos más utilizados, clasificados según la función de los modelos:

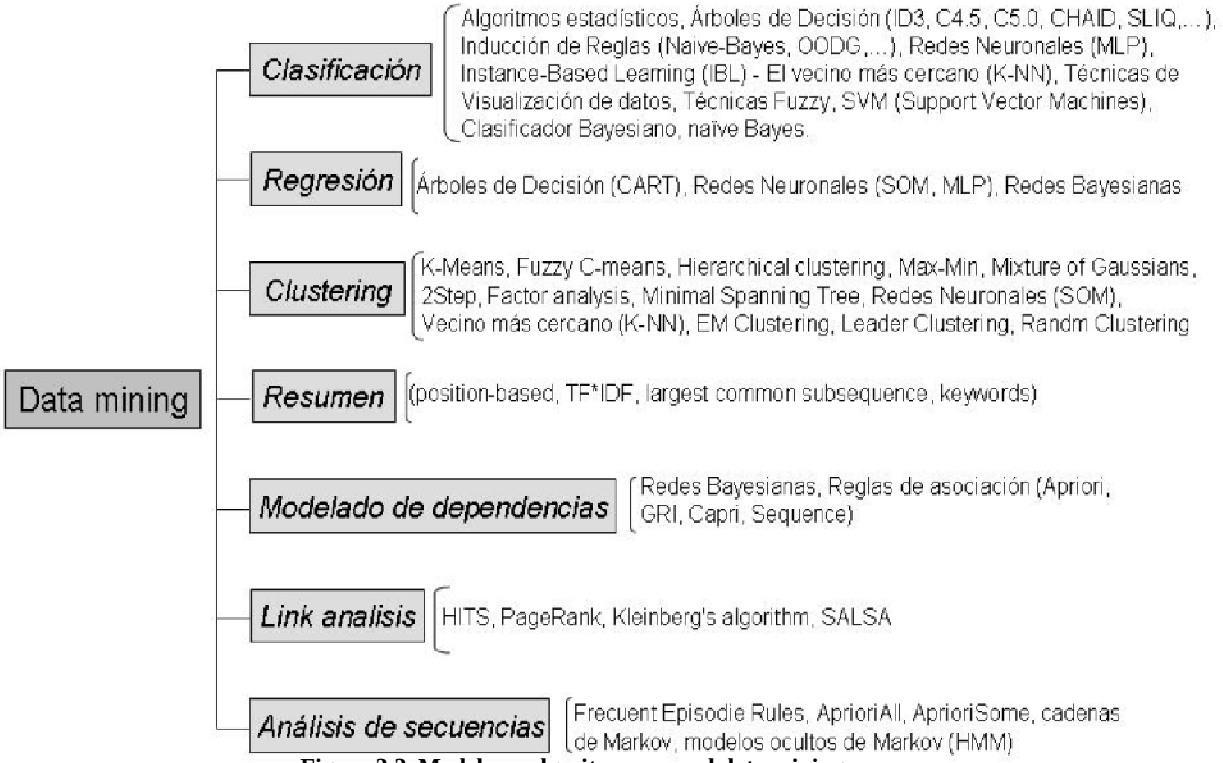


Figura 3.2 Modelos y algoritmos para el data mining

Típicamente, cuanto más complejos sean los modelos mejor se ajustarán a los datos, pero también será más complicada su comprensión y fiabilidad.

Hay que ser consciente de que no existe el mejor método de data mining; se dice que la elección del algoritmo para cada aplicación en particular tiene algo de arte (o quizás prueba-y- error).

Hay muchos métodos de hacer minería de datos. Una manera de clasificarlos es por las técnicas aplicadas. En este caso, pueden ser:

Sin algoritmo de aprendizaje:

- Consultas (SQL)
- Procesamiento analítico en línea (OLAP)
- Análisis estadístico (Correlación, regresiones,...)

Con algoritmo de aprendizaje:

- Redes neuronales y algoritmos genéticos
- Inducción de árboles y reglas

Nuevos algoritmos:

- Inducción de reglas de asociación
- Inducción de clasificadores bayesianos

Capítulo 4

Aprendizaje Automático y los Sistemas de Detección

Desde 1998 hasta ahora, numerosas investigaciones se centran en la aplicación del data mining en el campo de la detección de intrusos. Se han utilizado gran cantidad de algoritmos para muchas de las funciones existentes en el data mining, principalmente para la clasificación (aprendizaje supervisado) y clustering (no supervisado), pero también se pueden encontrar otras aplicaciones.

4.1 Aprendizaje supervisado Vs. aprendizaje no supervisado

Una característica principal dentro del data mining es el paradigma de aprendizaje de los sistemas. En el aprendizaje supervisado, se realiza la estimación basada en un conjunto de entrenamiento, del cual se conoce la clase a la que pertenece. Dicha estimación se extrae mediante la utilización de diversos algoritmos sobre un conjunto de registros de ataques previamente etiquetados. Se han utilizado una gran variedad de algoritmos, tanto por separado como en conjunto, para así poder aprovechar las capacidades de cada uno. Ejemplos de técnicas de este tipo son los árboles de decisión, reglas de asociación, etc.

En cambio, en el aprendizaje no supervisado, llamado también agrupación o *clustering*, el sistema de clasificación de patrones debe diseñarse partiendo de un conjunto de patrones de entrenamiento *{yi* ; j = 1,2,...,N} para los cuales no conocemos sus etiquetas de clase y,-. Estas situaciones se presentan cuando no disponemos del conocimiento de un experto o bien cuando el etiquetado de cada muestra individual es impracticable. Dentro de este tipo de aprendizaje destacan algoritmos de redes neuronales, el vecino más cercano, k-means, etc.

De todos modos, se pueden encontrar algoritmos o técnicas de aprendizaje automático que tienen la capacidad de aprender de ambas maneras, tanto supervisada como no supervisada (redes bayesianas, redes neuronales, vecino más cercano,...). En el campo de la detección de intrusos, se han realizado un gran número de proyectos que involucran a ambos tipos. En el siguiente estudio del estado del arte, se hace mayor hincapié en la técnica o algoritmo utilizado más que al tipo de aprendizaje al que pertenecen, ya que de lo contrario, a la hora de abordar técnicas utilizadas para la detección de intrusos como por ejemplo las redes neuronales, sería necesario un apartado para cada tipo de aprendizaje.

4.2 Reglas de asociación

Las reglas de asociación han sido ampliamente utilizadas para la detección de intrusos. Su objetivo es el de derivar correlaciones de varias características en una tabla de una base de datos, dada la evidencia de que frecuentemente los datos obtenidos de ejecuciones de programas y actividades de usuarios muestran correlaciones. Mediante las reglas de asociación se pueden encontrar relaciones internas entre datos de una misma conexión. Lee y Stolfo presentan sus algoritmos

de reglas de asociación y episodios frecuentes (para descubrir un conjunto de eventos que ocurrían en una ventana temporal) utilizados para calcular un conjunto de patrones a partir de datos de auditoria. Daniel Barbará y su grupo también hacen uso de esta técnica en su proyecto ADAM pero como método de detección de anomalías.

ADAM utiliza esta técnica de data mining para encontrar eventos anómalos en datos de conexiones de tráfico de red para luego clasificar dichos eventos o reglas de asociación inesperadas en instancias normales o anómalas. MINDS de la Universidad de Minnesota también utiliza reglas de asociación para caracterizar las anomalías detectadas, y ayudar en la creación de nuevas firmas y modelos de ataques emergentes. En IDDM (Intrusion Detection Data Mining) realizado por el departamento de defensa australiano, también incorporan un algoritmo de reglas de asociación a los agentes software. Finalmente, el trabajo realizado por min. Qin y Kai Hwang también está basado en la utilización de reglas de asociación y episodios frecuentes para la detección de anomalías.

4.3 Inducción de reglas

En 1990, Teng, Chen y Lu propusieron un método para descubrir patrones secuenciales temporales en una secuencia de eventos. De este modo, con el sistema Time-based Inductive Machine (TIM), se podían aprender patrones secuenciales para detectar intrusos.

La herramienta o algoritmo RIPPER, es un programa utilizado para el aprendizaje o inducción de conjuntos de reglas sobre un conjunto de datos previamente etiquetados (normal/intrusión). Ha demostrado ser más efectivo que el algoritmo de árboles de decisión C4.5 para datos con gran cantidad de ruido. Además genera un tipo de reglas fáciles de entender y de traducir a un lenguaje como Prolog. El sistema se compone de un conjunto de reglas de asociación y patrones de episodios frecuentes que pueden ser aplicados tanto a eventos de seguridad como a conjuntos de datos de tráfico de red. Wenkee Lee y Sal Stolfo fueron los que propusieron el uso de RIPPER para la detección de intrusos con éxito en su proyecto JAM, en el que proponen un innovador método para crear modelos de IDS y reglas de forma automática. Helmer realizo un trabajo parecido utilizando un conjunto más reducido de características. Otros en utilizar RIPPER para crear reglas de inducción fueron Warrender.

4.4 Vecinos más cercanos

Las técnicas de vecinos más cercanos *(NN, Nearest Neighbours)* basan su criterio de aprendizaje en la hipótesis de que los miembros de una población suelen compartir propiedades y características con los individuos que los rodean, de modo que es posible obtener información descriptiva de un individuo mediante la observación de sus vecinos más cercanos. Los fundamentos de la clasificación por vecindad fueron establecidos por E. Fix y J. L. Hodges a principio de los años 50. Sin embargo, no fue hasta 1967 cuando T. M. Cover y P. E. Hart enuncian formalmente la regla del vecino más cercano y la desarrollan como herramienta de clasificación de patrones. Desde entonces, este algoritmo se ha convertido en uno de los métodos de

clasificación más usados. La regla de clasificación *NN* se resume básicamente en el siguiente enunciado y se muestra con la ecuación 4.1:

Sea $\varepsilon = \{e_1,..., e_m\}$ un conjunto de datos con *m* ejemplos etiquetados, donde cada ejemplo *ej* contiene *n* atributos $(X_{j1}, ... ,Xj_n)$, pertenecientes al espacio métrico *X*, y una clase $y_1 \in \{y_1,,... ,y_k\}$ La clasificación de un nuevo ejemplo *e'* cumple que:

Donde:

$$e' \dashv y_l \Leftrightarrow \forall j \neq i \cdot d(e', e_i) < d(e', e_j) \qquad \textbf{(4.1)}$$

e' ← y_1 indica la asignación de la etiqueta de clase *yi* al ejemplo *e'*; y *d* expresa una distancia definida en el espacio *n*-dimensional *X*.

Así, un ejemplo es etiquetado con la clase de su vecino más cercano según la métrica definida por la distancia *d*. La elección de esta métrica es primordial, ya que determina qué significa más cercano. La aplicación de métricas distintas sobre un mismo conjunto de entrenamiento puede producir resultados diferentes. Sin embargo, no existe una definición previa que indique si una métrica es buena o no. Esto implica que es el experto quien debe seleccionar la medida de distancia más adecuada. La regla *NN* puede generalizarse calculando los *k* vecinos más cercanos y asignando la clase mayoritaria entre esos vecinos. Tal generalización se denomina *k-NN*. Este algoritmo necesita la especificación a priori de *k*, que determina el número de vecinos que se tendrán en cuenta para la predicción.

Al igual que la métrica, la selección de un *k* adecuado es un aspecto determinante. El problema de la elección del *k* ha sido ampliamente estudiado en la bibliografía. Existen diversos métodos para la estimación de *k*]. Otros autores han abordado el problema incorporando pesos a los distintos vecinos para mitigar los efectos de la elección de un *k* inadecuado. Otras alternativas intentan determinar el comportamiento de *k* en el espacio de características para obtener un patrón que determine a priori cuál es el número de vecinos más adecuado para clasificar un ejemplo concreto dependiendo de los valores de sus atributos. En otro estudio, F. J. Ferrer desarrollan un algoritmo de clasificación *NN* no parametrizado que adapta localmente el valor *k*.

El algoritmo *k-NN* se engloba dentro de las denominadas técnicas de aprendizaje perezoso *{lazy learning)*, ya que no genera una estructura de conocimiento que modele la información inherente del conjunto de entrenamiento, sino que el propio conjunto de datos representa el modelo.

Cada vez que se necesita clasificar un nuevo ejemplo, el algoritmo recorre el conjunto de entrenamiento para obtener los *k* vecinos y predecir su clase. Esto hace que el algoritmo sea computacionalmente costoso tanto en tiempo, ya que necesita recorrer los ejemplos en cada predicción, como en espacio, por la necesidad de mantener almacenado todo el conjunto de entrenamiento. Pese a los numerosos inconvenientes respecto a la eficiencia (coste computacional) y la eficacia (elección de la métrica y el *k* adecuados), *k-NN* tiene en general un buen comportamiento. Cover y Hart demostraron que, cuando el número de ejemplos tiende a infinito, el error asintótico de *NN* está acotado superiormente por el doble del error de Bayes (óptimo).

4.5 Árboles de Decisión

Los árboles de decisión, son una de las formas más sencillas de representación del conocimiento adquirido. Dentro de los sistemas basados en árboles de decisión, habitualmente denominados *TDIDT (Top Down Induction of Decisión Trees),* se pueden destacar dos familias o grupos: la familia *ID3,* cuyos máximos representantes son el propio algoritmo *ID3* propuesto por Quinlan y el sistema CLS de Hunt et; y la familia de árboles de regresión, cuyo exponente más significativo es CART, desarrollado por Breiman. Los *TDIDT* se caracterizan por utilizar una estrategia de divide y vencerás descendente, es decir, partiendo de los descriptores hacia los ejemplos, dividen el conjunto de datos en subconjuntos siguiendo un determinado criterio de división. A medida que el algoritmo avanza, el árbol crece y los subconjuntos de ejemplos son menos numerosos. *ID3* puede considerarse como una versión preliminar de *C4-5,* el cual resuelve algunos inconvenientes de su antecesor sobre el uso de atributos continuos, el tratamiento de valores ausentes y el proceso de poda. De los sistemas *TDIDT,* los pertenecientes a la familia *ID3* son los más referenciados en el campo del aprendizaje, por lo que serán expuestos con más detalle a continuación.

4.6 Naive Bayes

Los principios estadísticos Bayesianos han resultado de gran utilidad para los IDS. El resultado de la aplicación de sistemas Bayesianos se presenta en forma de relaciones de probabilidad condicional, en lugar de reglas o firmas. Las redes Bayesianas son unas herramientas poderosas como modelos de decisión y razonamiento bajo incertidumbre.

El Clasificador Bayesiano naive, se utiliza cuando queremos clasificar una instancia descrita por un conjunto de atributos en un conjunto finito de clases. Naive Bayes es un tipo simple de redes Bayesianas particularmente eficientes en tareas de inferencia.

Naive-Bayes es una técnica de clasificación descriptiva y predictiva basada en la teoría de la probabilidad del análisis de T. Bayes, que data de 1763. Esta teoría supone un tamaño de la muestra asintóticamente infinito e independencia estadística entre variables independientes, refiriéndose en nuestro caso a los atributos, no a la clase. Con estas condiciones, se puede calcular las distribuciones de probabilidad de cada clase para establecer la relación entre los atributos (variables independientes) y la clase (variable dependiente).

Concretamente, dado el ejemplo $e = (x_{1,\ldots} xn)$, donde *Xi* es el valor observado para el í-ésimo atributo, la probabilidad a posteriori de que ocurra la clase *yi* teniendo *k* valores posibles $\{y_i,\ldots, y_k\}$, viene dada por la regla de Bayes, véase ecuación 4.2.

$$P(y_l|x_1,\ldots,x_n) = \frac{P(y_l)\prod_{i=1}^{n} P(x_i|y_l)}{P(x_1,\ldots,x_n)} \qquad \textbf{(4.2)}$$

donde *P(yi)* es la proporción de la clase *yi* en el conjunto de datos; e igualmente, *P(xi\yi)* se estima a partir de la proporción de ejemplos con valor *Xi* cuya clase es y_i. Como podemos deducir, el cálculo de *P(xi\yi)* obliga a que los valores x_i sean

discretos, por lo que si existe algún atributo continuo, éste debe ser discretizado previamente. Aplicando la ecuación 4.3, la clasificación de un nuevo ejemplo *e* se lleva a cabo calculando las probabilidades condicionadas de cada clase y escogiendo aquella con mayor probabilidad. Formalmente, si *Dom{Y)* = *{y\,... ,y*$_k$*}* es el conjunto de clases existentes, el ejemplo *e* será clasificado con aquella clase *y¡* que satisface la expresión:

$$\forall j \neq i \cdot P(y_i|x_1, \ldots, x_n) > P(y_j|x_1, \ldots, x_n) \quad \textbf{(4.3)}$$

Como se puede observar, el clasificador bayesiano es un método sencillo y rápido. Además, puede demostrarse teóricamente que maximiza la exactitud de la predicción de manera óptima. Sin embargo, la suposición de independencia estadística de las variables es una limitación importante, ya que este hecho es relativamente infrecuente.

4.7 ID3

El método de clasificación experimental ID3 *(Induction Decisión Trees),* desarrollado por J. R. Quinlan, genera un árbol de decisión paralelo de forma recursiva, aplicando un criterio de división basado en el concepto de medida de la información de Shannon. Cada nodo interno de dicho árbol contiene un test sobre uno de los atributos, de cuyo valor dependerá el camino a seguir para clasificar un ejemplo, y cada hoja contiene una etiqueta de clase. Así, la clasificación de un ejemplo se lleva a cabo recorriendo el árbol desde la raíz hasta una de las hojas que determinará la clase del mismo. Inicialmente, el algoritmo toma todo el conjunto de datos £. Si todos los ejemplos pertenecen a una misma clase, el proceso finaliza, insertando un nodo hoja con dicha clase. En caso contrario, se selecciona aquel atributo *X¿* que mejor divide el conjunto de datos y se inserta un nodo con dicho atributo para establecer un test. Una vez creado el nodo, para cada valor distinto *Xi*$_V$ del atributo *X¿,* se traza un arco y se invoca recursivamente al algoritmo para generar el subárbol que clasifica los ejemplos de £ que cumplen que *X¿* = *x*$_{iv}$. Dicha invocación es realizada sin tener en cuenta el atributo *X¿* y substrayendo del conjunto de datos £ todos aquellos ejemplos donde *X¿* ^ *Xi*$_V$. El proceso se detiene cuando todas las instancias de un conjunto pertenecen a la misma clase. *ID3* utiliza una propiedad estadística denominada *ganancia de información* como heurística de selección de atributos para fijar un test. Esta propiedad no es más que la reducción esperada de la entropía (desorden) de los datos al conocer el valor de un atributo. Así, el atributo *X¿* seleccionado para determinar la división será aquel que mayor ganancia obtenga respecto al conjunto £, según la ecuación 4.4:

$$Ganancia(\mathcal{E}, X_i) = Ent(\mathcal{E}) - \sum_{v=1}^{|X_i|} \frac{|\mathcal{E}(x_{iv})|}{|\mathcal{E}|} \times Ent(\mathcal{E}(x_{iv})) \quad \textbf{(4.4)}$$

donde |*Xi*| es el número de valores distintos de del atributo *Xi*; *£(X*$_{IV}$*)* es el subconjunto de £ para el cual *Xi* = *X*$_{iv}$, siendo |*ε*(*x*$_{iv}$)| su cardinal; \ε\ es el número total de ejemplos; y Ent(-) es la entropía, definida a continuación.

La entropía es la medida del desorden de un sistema mediante la incertidumbre existente ante un conjunto de casos, del cual se espera uno sólo. La ecuación 4.5 muestra la medida de desorden. Sea £ un conjunto de datos etiquetados con clases del conjunto Dom{Y) = {y_1,... ,yk} y frec(yi,ε) el número de ejemplos de ε con clase y¡. Entonces se define la entropía del conjunto ε como:

$$Ent(\mathcal{E}) = -\sum_{l=1}^{k} \frac{frec(y_l, \mathcal{E})}{|\mathcal{E}|} \times \log_2 \left(\frac{frec(y_l, \mathcal{E})}{|\mathcal{E}|} \right) \qquad \textbf{(4.5)}$$

Donde es la probabilidad de que se dé un ejemplo con clase yi, y $\log_2$ es la información que transmite un ejemplo de clase y¡. La entropía es máxima cuando todas las cases presentan la misma proporción.

Pese a su simplicidad y bajo coste computacional, *ID3* presenta inconvenientes importantes, algunos de los cuales son corregidos por su sucesor *C4-5*. Los más evidentes son la incapacidad para trabajar con atributos continuos y tratar valores ausentes. Sin embargo, presenta una serie de problemas que afectan directamente a la precisión del árbol generado. En primer lugar, la heurística usada para establecer los test es propensa a seleccionar aquellos atributos con mayor número de valores distintos, ya que a mayor número de particiones, la entropía de cada subconjunto tiende a ser menor.

En segundo lugar, *ID3* resulta muy vulnerable a la presencia de ruido e inconsistencia en los datos, lo cual ocasiona la generación de *hojas muertas* que clasifican ejemplos de más de una clase. Por último, la limitada capacidad de generalización del algoritmo provoca la aparición de *hojas vacías*, que no clasifican ningún ejemplo del conjunto de entrenamiento y, por lo tanto, no se les asigna etiqueta de clase. Esto implica que no se podrán realizar predicciones sobre aquellos ejemplos incluidos en las zonas del espacio cubiertas por hojas vacías por no aparecer en el conjunto de entrenamiento.

Por otra parte, el algoritmo obliga a que todos los ejemplos sean clasificados correctamente. Esto, unido a los problemas de generalización y ruido, hace que *ID3* produzca árboles de mucha profundidad sin que esto beneficie a la precisión de los mismos. Quinlan propuso como solución un método de poda para reducir el error y el tamaño de los árboles. Dicho método sustituía un subárbol completo por una hoja etiquetada con la case mayoritaria del subárbol si ésta sustitución mejoraba o al menos iguala la clasificación original.

4.8 C4.5

El algoritmo C4-5 fue propuesto por Quinlan a finales de los años 80 para mejorar las carencias de su predecesor ID3. Desde entonces, ha sido uno de los sistemas clasificadores más referenciados en la bibliografía, principalmente debido a su extremada robustez en un gran número de dominios y su bajo coste computacional. C4-5 introduce principalmente las siguientes mejoras:

Trata eficazmente los valores desconocidos calculando la ganancia de información para los valores presentes.

Maneja los atributos continuos, aplicando una discretización previa.

Corrige la tendencia de ID3 a seleccionar los atributos con muchos valores distintos para establecer los test cambiando el criterio de división.

C4-5 produce un árbol de decisión similar al de ID3, con la salvedad de que puede incluir condiciones sobre atributos continuos. Así, los nodos internos pueden contener dos tipos de test según el dominio del atributo seleccionado para la partición. Si el atributo Xi es discreto, la representación es similar a la de ID3, presentando un test con una condición de salida (rama Xi = xiv) por cada valor xiV diferente del atributo.

Por contra, si el atributo Xi es continuo, el test presenta dos únicas salidas, Xi < Z y Xi > Z, que comparan el valor de Xi con el umbral Z. Para calcular Z, se aplica un método similar , el cual ordena el conjunto de t valores distintos del atributo Xi presentes en el conjunto de entrenamiento, obteniendo el conjunto de valores {xi1,xi2,….,xit}.que se muestran en la ecuación 4.6:

$$\frac{x_{iv}+x_{i(v+1)}}{2} \qquad \textbf{(4.6)}$$

Cada par de valores consecutivos aporta un posible umbral Z= , teniendo en total t — 1 umbrales, donde t es como mucho igual al número de ejemplos. Una vez calculados los umbrales, CJh5 selecciona aquel que maximiza el criterio de separación. Como se mencionó anteriormente, el criterio de maximización de la ganancia de información usado en ID3 produce un sesgo hacia los atributos que presentan muchos valores distintos. C4-5 resuelve este problema usando la razón de ganancia {gain ratio) como criterio de separación a la hora de establecer un test. Esta medida tiene en cuenta tanto la ganancia de información como las probabilidades de los distintos valores del atributo. Dichas probabilidades son recogidas mediante la denominada información de separación (split information), que no es más que la entropía del conjunto de datos £ respecto a los valores del atributo Xi en consideración, siendo calculada mediante la ecuación 4.7:

$$InformacionDeSeparacion(\mathcal{E}, X_i) = -\sum_{v=1}^{|X_i|} \frac{|\mathcal{E}(x_{iv})|}{|\mathcal{E}|} \times \log_2 \left(\frac{|\mathcal{E}(x_{iv})|}{|\mathcal{E}|} \right) \qquad \textbf{(4.7)}$$

donde |Xi| es el número de valores distintos del atributo Xi; ε(xiv) es el subconjunto de £ para el cual Xi = XiV, siendo |ε(xiv)| su cardinal; y \ε\ es el número total de ejemplos. La información de separación simboliza la información potencial que representa dividir el conjunto de datos, y es usada para compensar la menor ganancia de aquellos test con pocas salidas. Con ello, tal y como muestra la ecuación 4.8, la razón de ganancia es calculada como el cociente entre la ganancia de información (ecuación 4.4) y la información de separación (ecuación 4.7). Tal cociente expresa la proporción de información útil generada por la división.

$$RazonDeGanancia(\mathcal{E}, X_i) = \frac{Ganancia(\mathcal{E}, X_i)}{InformacionDeSeparacion(\mathcal{E}, X_i)} \qquad \textbf{(4.8)}$$

C4.5 maximiza este criterio de separación, premiando así a aquellos atributos que, aun teniendo una ganancia de información menor, disponen también de menor número de valores para llevar a cabo la clasificación. Sin embargo, si el test incluye pocos valores, la información de separación puede ser cercana a cero, y por tanto el

cociente sería inestable. Para evitar tal situación, el criterio selecciona un test que maximice la razón de ganancia pero obligando a que la ganancia del mismo sea al menos igual a la ganancia media de todos los test examinados. C4-5 ha resultado ser un sistema muy efectivo en la práctica, capaz de ofrecer una representación relativamente simple de los resultados con un bajo coste computacional. En concreto, para un conjunto de datos con m ejemplos y n atributos, el coste medio de construcción del árbol es de Q(mnlog2m), mientras que la complejidad del proceso de poda es de Q(m(log2m)2). Por contra, el algoritmo presenta también dos inconvenientes importantes derivados de la representación del conocimiento que obtiene y la metodología seguida para ello:

La representación mediante árboles de decisión paralelos puede provocar que zonas contiguas en el espacio no puedan ser unidas para simplificar la regla. Esto hace que el árbol tienda a crecer sustancialmente en aplicaciones reales, complicando la compresión del mismo.

La estrategia seguida establece en cada paso una única frontera de decisión para un solo atributo, sin posibilidad de reajustar el modelo en pasos posteriores. Es decir, C4-5 establece en un momento dado una condición sobre un atributo porque en ese instante entiende que es la mejor, sin tener en consideración que en el proceso posterior de establecer condiciones sobre los demás atributos, esa primera opción pudiera no ser la mejor.

4.9 Multiclasificadores

Los multiclasificadores son conjuntos de clasificadores diferentes que realizan predicciones que se fusionan y se obtiene como resultado la combinación de cada una de ellas. El hecho o la idea de combinación hace que los multiclasificadores sean citados en la literatura a través de distintos términos, entre ellos: métodos de ensamble, modelos múltiples, sistemas de múltiples clasificadores, combinación de clasificadores, integración de clasificadores, mezcla de expertos, comité de decisión, etc.

Se dividen en dos grupos: los métodos de ensamble o ensamblaje y los métodos híbridos. Los primeros se refieren a aquellos conjuntos de modelos que se combinan para crear un nuevo modelo, empleando para ello la misma técnica de aprendizaje.
En el caso de los segundos, son combinaciones de algoritmos de aprendizaje que crean a su vez nuevas técnicas de aprendizaje híbridas. Los multiclasificadores se distinguen unos de otros atendiendo a diversas características. Entre ellas podemos citar: el número de clasificadores individuales acoplados, el tipo de cada clasificador (redes neuronales, árboles de decisión, vecino más cercano, etc.), las características de los subconjuntos usados por cada clasificador del conjunto, la agregación de las decisiones particulares (voto mayoritario, asignación de pesos, subespacio de mejor comportamiento, funciones de promedio, máximo, mínimo, producto, etc.) y el tamaño y la naturaleza de los conjuntos de datos de entrenamiento para los clasificadores.

La evaluación y comparación entre los distintos multiclasificadores se establece a través de indicadores de rendimiento que incluyen el grado de generalización, aprendizaje, clasificación correcta e incorrecta y el tiempo real de ejecución.

4.9.1 Bagging

Bagging es un método propuesto por Breiman y está basado en los conceptos de bootstrapping y agregación, de esta forma se reincorporan los beneficios de ambos y se le da nombre al método (Bootstrap AGGregatING). Boostrapping se basa en la generación de conjuntos de entrenamientos aleatorios con reemplazamiento. Una muestra bootstrap se genera al muestrear uniformemente instancias del conjunto de entrenamiento de manera aleatoria esta se muestra en la figura 4.1. De modo que se crean tantas muestras bootstrap como clasificadores existan, de ahí que cada clasificador se entrena con una réplica bootstrap.

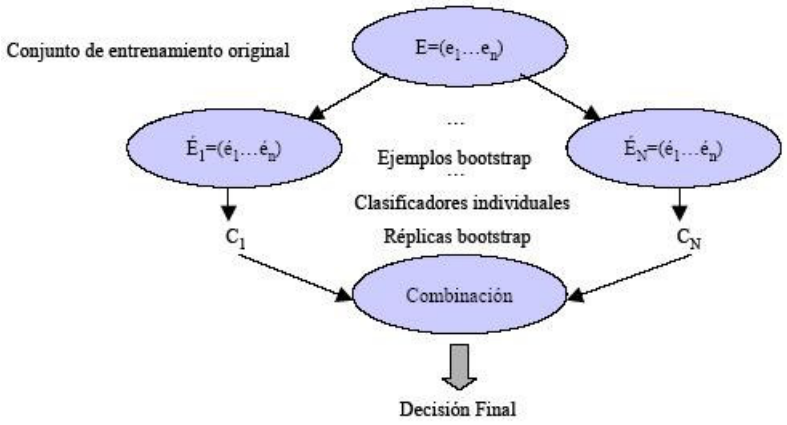


Figura 4.1 Esquema del método bagging

El método consiste en que los clasificadores individuales acoplados durante K iteraciones extraigan cada vez una muestra aleatoria del subconjunto É de n ejemplos con reemplazamiento a partir del conjunto original de ejemplos (pueden haber ejemplos repetidos); el subconjunto extraído aprende un modelo (técnica de aprendizaje a partir de una evidencia). Para clasificar un ejemplo e, se predice la clase de ese ejemplo para cada clasificador y se selecciona la clase con mayor voto.

4.10 Comités de validación cruzada (Cross-Validated Committees)

La idea es dividir los datos de entrenamiento en varios conjuntos. Los conjuntos no tienen que ser necesariamente mutuamente exclusivos, ellos pueden compartir parte del conjunto (solaparse). Esta idea es similar a los métodos de remuestreo como la validación cruzada (crossvalidation) que es utilizado en estadística para estimar el error de un predictor a partir de los datos disponibles. En el contexto de comité, esta técnica se reforma para construir conjuntos de entrenamiento diferentes del conjunto original, de ahí el nombre de comités de validación cruzada (CVC). El algoritmo primero genera réplicas a partir del conjunto de entrenamiento original mediante la exclusión de un fragmento de los datos. Se denota D como los datos originales y D^{-v} denota los datos con el subconjunto v excluido. El procedimiento consiste en ir rotando para que cada elemento esté por lo menos una vez como fragmento excluido. Ver figura 4.2.

Se generan réplicas $D_1^{-v1},...,D_k^{vk}$ y se entrena a cada clasificador que forma parte del comité.

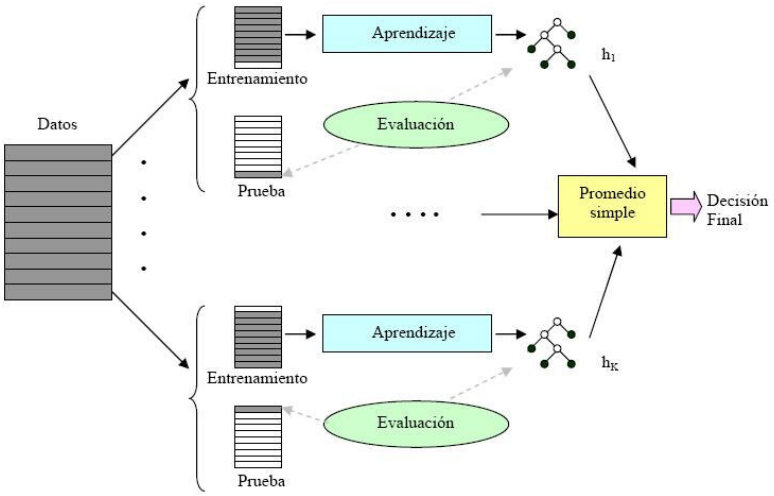


Figura 4.2 Algoritmo de Comités de Validación Cruzada

Los estimadores de error se basan en calcular la proporción de las instancias incorrectamente etiquetadas por el clasificador esto se muestra en la ecuación 4.9. Una vez construido el clasificador d y dado una instancia (X ,c):

$$\Delta(X_i,c_i)=\begin{cases}1;\ \text{si } d(X_i)\neq c_i \text{ (error)}\\ 0;\text{no } d(X_i)=c_i \text{ (acierto)}\end{cases} \qquad \textbf{(4.9)}$$

Para todo k, k=1,2,..., K, se construye un clasificador d_k , usando D−D_k como conjunto de aprendizaje. Como ninguna de las instancias de D_k se ha usado para construir d_k , el estimador mediante el conjunto de prueba de d_k es $R(d_k)$:

$$R(d_k)=\frac{1}{|D_k|}\sum_{(X_i,c_i)\in D_k} D(X_i,c_i) \qquad \textbf{(4.10)}$$

Donde Δ se evalúa sobre d_k y D_k es el conjunto de prueba. Al finalizar se obtiene K clasificadores, d_k con sus correspondientes estimaciones de error $R(d_k)$.

Posteriormente, se usa el mismo procedimiento para construir el clasificador *d* con todas las instancias de D. Como cada d_k se construye con D-D , si *K* es grande,

$$|D-D_k|=N(1-\frac{1}{k})\approx|T|. \qquad \textbf{(4.11)}$$

Este procedimiento es estable (todos los clasificadores d_k tienen una tasa de error aproximadamente igual a la del clasificador d). Cuando *K=N*, el estimador por validación cruzada con *N* (tamaño del conjunto de aprendizaje) conjuntos se conoce como el estimador que deja uno fuera (del inglés leave-one-out).

Para cada *n, n=1,2,...,N* el n−esimo ejemplo es descartado y el clasificador se construye utilizando los restantes *N−1* ejemplos. Entonces, el ejemplo descartado se usa para prueba y se estima el error mediante la ecuación 4.12:

K

$$R^{cv}(d) = \frac{1}{K}\sum_{k=1}^{K} R(d_k) \qquad \textbf{(4.12)}$$

El algoritmo:

1. *Dividir los datos en v-fragmentos* $d_1,...,d_v$

2. Leave-one-out (dejar un fragmento fuera) d_k y entrenar la red h_k con el resto de los datos ($D - d_k$)

3. Usar d_k como un criterio de parada de la validación, si es necesario

4. Construir un comité (unir los resultados de los clasificadores) a partir de las redes usando un procedimiento de promedio simple.

Este algoritmo requiere de un gran esfuerzo computacional, ya que todos los

ejemplos de D se usan para construir los d_k , y cada uno de ellos se usa exactamente una vez para prueba. No obstante, es adecuado para subconjuntos de pequeño tamaño.

4.11 Métodos de validación

La estrategia para validar un sistema de aprendizaje depende esencialmente de la manera en que dicha división es realizada. Algunos autores utilizan el mismo conjunto para el entrenamiento y el test, lo que produce una tasa de error casi siempre menor a la real y a menudo una estimación demasiado optimista. Para que la estimación sea válida, los conjuntos de entrenamiento y test deben ser independientes, o al menos diferentes. Los métodos de validación más destacados son:

- **Validación cruzada**. La validación cruzada con k conjuntos (k-fold cross validation) es atribuida a M. Stone. Consiste en dividir los datos en k subconjuntos de ejemplos con aproximadamente igual tamaño y evaluar el sistema k veces. En cada evaluación, se deja uno de los subconjuntos para el test y se entrena el sistema con los k−1 restantes. Así, el error estimado es la media de las k tasas obtenidas. A partir de este método de validación cruzada básico, se han desarrollado variantes para aproximar mejor la tasa de error.
- **Validación cruzada completa.** El método de validación cruzada completa (complete k−fold cross validation) realiza una exploración completa de todos las posibles combinaciones de N ejemplos en el conjunto de test, K donde N es el número total de ejemplos, dejando el resto para el entrenamiento. En concreto se llevan a cabo evaluaciones, lo cual resulta extremadamente costoso computacionalmente.
- **Validación cruzada estratificada.** Una variante del método básico, denominada validación cruzada estratificada (stratified k-fold cross validation), distribuye los ejemplos intentando mantener la misma proporción de instancias de cada clase en el conjunto de entrenamiento y en el de test. Éste es quizá el método de validación a más recomendable, ya que mantiene para las evaluaciones la misma proporción de clases del conjunto

total de datos, además de no aumentar significativamente el coste computacional respecto al método original.

- **Validación dejando uno fuera**. Este método es la validación cruzada llevada al extremo, es decir, tomando k igual al número de ejemplos (N) del conjunto de datos. Así, el clasificador entrena con N−1 ejemplos, dejando uno fuera para realizar el test, de ahí su nombre (leave–one–out). Además de la elevada varianza de la tasa de error obtenida, el mayor inconveniente de este método es su alto coste computacional, por lo que no es recomendable su uso con más de 100 ejemplos.
- **Split Sample**. Split sample es un tipo de validación no cruzada donde sólo existe un conjunto de test sobre el cual se estima el error. El tamaño dicho conjunto suele rondar entre el 20% y 30% del conjunto original, empleándose el resto para entrenamiento. La selección de los ejemplos para cada conjunto es aleatoria, aunque es aconsejable asegurar la misma proporción de ejemplos de distintas clases en ambos.
- **Bootstrapping.** Existen varias formas de aplicar la validación bootstrapping o validación por secuencia. La más simple consiste en realizar un muestreo aleatorio con reemplazo en el conjunto de datos, copiando los ejemplos seleccionados en el conjunto de entrenamiento hasta que éste alcance el tamaño del original. El conjunto de test lo conforman aquellos ejemplos no incluidos en el entrenamiento. Aunque la tasa de error de este conjunto es un estimador del error real, lo habitual es repetir el proceso varias veces y calcular la media.

Capítulo 5

Experimentos

Este capitulo analiza los resultados obtenidos de las pruebas realizadas con la herramienta Weka que fue la que seleccione por que cuenta con los algoritmos estudiados en este trabajo, además de que cuenta con licencia GPL lo cual ha inducido para que sea una de las más utilizadas en el área de minería de datos en los últimos años.

Se mostrara una descripción de los datos de como están divididos y las clases que se utilizaron en cada uno de las pruebas. Se interpretaran los resultados obtenidos y se explicara la manera en la que se llevarán a cabo cada una de las pruebas.

Los experimentos se realizaron utilizando 3 métodos de validación, validación cruzada a 10 folds, Split Sample 66% y utilizando un conjunto de datos de entrenamiento. Para ellos se hicieron 2 pruebas con cada uno de estos métodos, el primero utilizando los 42 atributos del conjunto de datos y el segundo solo usando 8.

En estos experimentos se utilizaron los algoritmos: Árboles de decisión J48 y Random Forest, Naive Bayes, ZeroR , Ridor.

5.1 HERRAMIENTAS DE TRABAJO

5.1.1 WEKA

WEKA, acrónimo de Waikato Environment for Knowledge Analysis, es un entorno para experimentación de análisis de datos que permite aplicar, analizar y evaluar las técnicas más relevantes de análisis de datos, principalmente las provenientes del aprendizaje automático, sobre cualquier conjunto de datos del usuario. Es un software compuesto de un conjunto de librerías JAVA que ha sido desarrollado en la universidad de Waikato (Nueva Zelanda) bajo licencia GPL, lo cual ha impulsado que sea una de las suites más utilizadas en el área en los últimos años.

Para ello únicamente se requiere que los datos a analizar se almacenen con un cierto formato, conocido como ARFF (Attribute-Relation File Format).

5.2 Descripción de los datos.

El aprendizaje para un detector de intrusos consiste en construir un modelo predictivo (es decir, un clasificador) capaz de distinguir entre "malas" conexiones, llamadas intrusiones o ataques, y "buenas'"conexiones.

.

En 1998, la DARPA (Agencia de Proyectos de Investigación Avanzada de la Defensa) desarrollo y administro el Programa de Evaluación de detección de Intrusos. El objetivo era estudiar y evaluar la investigación en la detección de intrusiones que incluye un conjunto estándar de datos a ser auditados, que incluye una amplia variedad de intrusiones simuladas en una red militar. Para ellos se creo un ambiente simple de conexiones TCP para una red local que simulaba la de la Fuerza Aérea de EE.UU. Se opero la LAN como si se tratara de un verdadero entorno de la Fuerza Militar pero salpicado con múltiples ataques.

Los datos de entrenamiento eran cerca de 4 gigabytes que se almacenaron durante 7 semanas. Estos datos fueron transformados en cerca de 5 millones de registros de conexión.

Una conexión es una secuencia de paquetes TCP de inicio y finalización en algunas ocasiones bien definidas, entre las que los flujos de datos van hacia una dirección IP de origen a una dirección IP de destino, en algunos casos con protocolos bien definidos. Cada conexión tiene la etiqueta, ya sea como normal, o como un ataque. Cada registro de conexión se compone de alrededor de 100 bytes.

Los ataques se dividen en cuatro categorías principales:

DOS: denegación de servicio,

R2L: el acceso no autorizado desde una máquina remota.

U2R: el acceso no autorizado a los privilegios del superusuario (root).
PROBE: acceso de vigilancia y sondeo.

5.3 PREPARACIÓN DE LOS DATOS

5.3.1 Selección de los datos

Los datos fueron tomados de la página de la competición KDD 1999. La tarea de la competencia fue la de construir un detector de intrusos, un modelo predictivo capaz de distinguir entre "mala'" conexiones, llamadas intrusiones o ataques, y "buenas" conexiones. Esta base de datos contiene un conjunto estándar de datos a ser auditados, que incluye una amplia variedad de intrusiones simulada en un entorno de una red militar como se comento anteriormente.

Los archivos son:

kddcup.data: Datos de entrenamiento originales (743 MB).

kddcup.data_10_percent: Subconjunto del 10% de los datos de entrenamiento (75 MB).

kddcup.testdata.unlabeled: Datos de test originales sin etiquetar (430 MB).

kddcup.testdata.unlabeled_10_percent: Subconjunto del 10% de los datos de test sin etiquetar (45 MB).

Corrected: Subconjunto anterior con los datos correctamente etiquetados (ataques correspondientes especificados) (48 MB).

Se debe tener en cuenta que la herramienta WEKA tiene que manejar grandes conjuntos de datos y que al usar la máquina virtual de java se dispone de una memoria limitada. Por este motivo hay que tomar un conjunto de datos que sea lo bastante grande como para poder hacer un análisis que se considere lo suficientemente bueno, pero que a la vez sea lo necesariamente pequeño para que WEKA pueda manejarlo con cierta soltura sin llegar a colapsar el equipo.

Ateniéndose a estas circunstancias se descartan los conjuntos de datos originales, quedándose los subconjuntos del 10%.

Entre los datos de test etiquetados y sin etiquetar se han elegido directamente los etiquetados, pues con WEKA se puede hacer una predicción del tipo de ataque a la vez que se compara con el ataque real, proporcionando estadísticas de la calidad de la predicción y de sus errores.

De esta forma se dispone de un conjunto de 494021 instancias de datos de entrenamiento y 311029 instancias de datos corregidos, que siguen siendo demasiado grandes como para poder trabajar con ellos.

El conjunto que más limita es el de datos de entrenamiento, pues con él se tiene que construir el modelo de clasificación, lo que consume mucha memoria. El proceso de clasificación de los datos de test suele ser mucho más rápido.

Probando con diferentes tamaños de conjuntos de datos se han elegido finalmente un subconjunto de 20585 instancias de datos de entrenamiento y 10034 de test, pues con estos conjuntos se pueden utilizar casi todos los algoritmos de minería de datos.

La elección de estos sub-subconjuntos se ha hecho muestreando los datos (cogiendo por ejemplo uno de cada 24 en el caso de los de entrenamiento), para tener unas muestras heterogéneas.

5.3.2 Acondicionamiento al formato WEKA

Una vez que se han elegido los datos, se tienen que pasar a formato WEKA. Los datos de entrada sobre los que operarán los algoritmos, deben estar codificados en un formato denominado Attribute-Relation File Format (extensión “arff”).

El formato de un fichero arff sigue la estructura siguiente:

```
@relation NOMBRE_RELACION
@attribute at1 tipo
@attribute at2 {valor1, valor2, …}
    …
@attribute atN {valor1, valor2, …}
@data dato11,dato21...,datoN1
    …
      dato1M,dato2M...,datoNM
```

5.4 Atributos y clases

Los atributos pueden ser principalmente de dos tipos: numéricos de tipo real o entero (real o integer), y simbólicos (especificando los valores posibles que pueden tomar entre llaves).

En este caso, tenemos 42 atributos, los de tipo continuo se han puesto en WEKA como integer, los discretos boléanos (que toman valores 0 o 1) como simbólicos ({0,1}), y el resto de atributos discretos, con sus valores correspondiente.

Los datos se encuentran divididos en 23 clases que corresponden a los distintos tipos de ataque que se pueden presentar.

Los valores que pueden tomar estos atributos se muestran en la tabla 5.1:

Atributo	Valore
protocol_type	Tcp, udp, icmp
service	http, mtp, smtp, finger, domain, domain_u, auth, telnet, ftp, eco_i, ntp_u, ecr_i, other, private, pop_3, ftp_data, rje, time, link, remote_job, gopher, ssh, name, whois, login, imap4, daytime, ctf, nntp, shell, IRC, nnsp, http_443, exec, printer, icmp, efs, courier, uucp, klogin, kshell, echo, discard, systat, supdup, iso_tsap, hostnames, csnet_ns, pop_2, sunrpc, uucp_path, netbios_ns, netbios_ssn, netbios_dgm, sql_net, vmnet, bgp, Z39_50, ldap, netstat, urh_i, X11, urp_i, pm_dump, tftp_u, tim_i, red_i
flag	SF, S1, REJ, S2, S0, S3, RSTO, RSTR, RSTOS0, OTH, SH
clase (ataque)	back, buffer_overflow, ftp_write, guess_passwd, imap, ipsweep, land, loadmodule, multihop, neptune, nmap, normal, perl, phf, pod, portsweep, rootkit, satan, smurf, spy, teardrop, warezclient, warezmaster

Tabla 5.1 Posibles valores de los atributos simbólicos.

5.5 DISTRIBUCIÓN DE LOS DATOS DE ENTRENAMIENTO

Cuando se introducen los datos de entrenamiento en WEKA, la herramienta de visualización permite representar gráficas en 2D que relacionan atributos.

En la siguiente figura 5.1 se ve con qué frecuencia aparece cada tipo de ataque en los datos de entrenamiento.

Back	975
Buffer_overflow	5
ftp_write	8
Guess_passwd	53
Imap	12
Ipsqeep	94
Land	10
Loadmodule	4
Multihop	6
Neptuno	407
Nmap	40
Normal	17193
Perl	1
Phf	1
Pod	20
portsweep	40
Rootkit	5
Satan	794
Smurf	3695
Spy	2
Teardrop	103
Warezlient	15
warezmaster	15

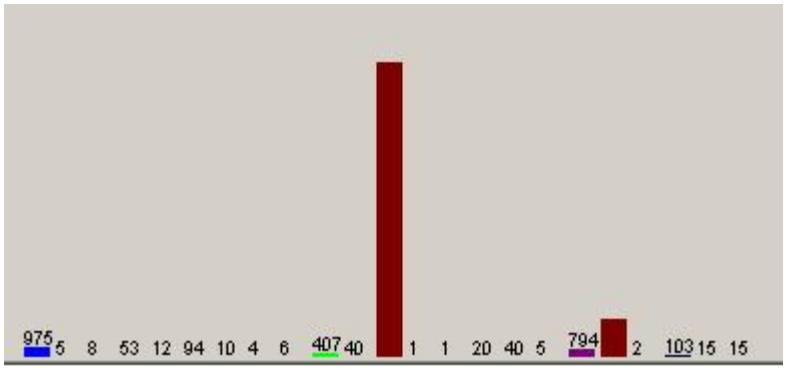


Figura 5.1.Frecuencia de aparición de cada ataque.

Como ya se dijo, los ataques de tipo R2L y U2R están contenidos dentro de un paquete de datos y suelen consistir en una única conexión. Se comprueba esto en la Figura, viendo que algunos de ellos, constan de una única aparición o muy pocas apariciones (como 'buffer_overflow', 'imap', 'loadmodule', 'phf' ,'perl' y'warezmaster').

Los ataques de denegación de servicio (DoS) aparecen casi todos en muchas ocasiones ('back' 975 veces, 'pod' 20 veces, 'teardrop' 103 veces o 'land' 10 veces), destacando los que se basan en inundación que sobresalen ampliamente del resto, con las 407 apariciones de 'neptune' y 'smurf' con 3695 conexiones, siendo el ataque más frecuente de todos.

Los ataque de sondeo (probing) tienen varias ocurrencias cada uno: 'ipsweep' (94), 'nmap' (40), 'portsweep' (40), 'Satán' (794).

Finalmente se ve que las conexiones buenas o normales son mucho mas frecuentes que las demás con 28956 apariciones.

Observando la figura 5.2 se puede ver que el tipo de ataque depende mucho del tipo de protocolo que se use. Por ejemplo todos los ataques 'smurf' usan 'icmp', los 'teardrop' 'udp' y los de tipo 'back', 'tcp', por lo que este atributo es muy importante para aumentar la calidad de la predicción.

Con el atributo 'service' pasa algo muy parecido.

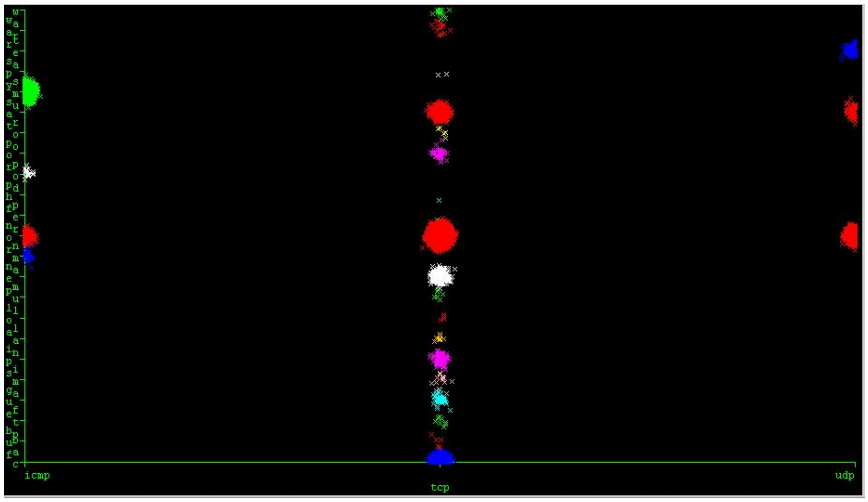


Figura 5.2. Gráfica protocol_type - class.

En figura 5.3 se observa a su vez cómo casi todos los ataques 'teardrop' tienen un número de fragmentos erróneos superior al resto de ataques.

Figura 5.3 Gráfica wrong_fragment - class.

Con los atributos especiales también se ve cómo se distribuyen los datos. Todos los ataques 'warezclient' y 'back' tienen un 'logged_in' exitoso que se muestra en la figura 5.4.

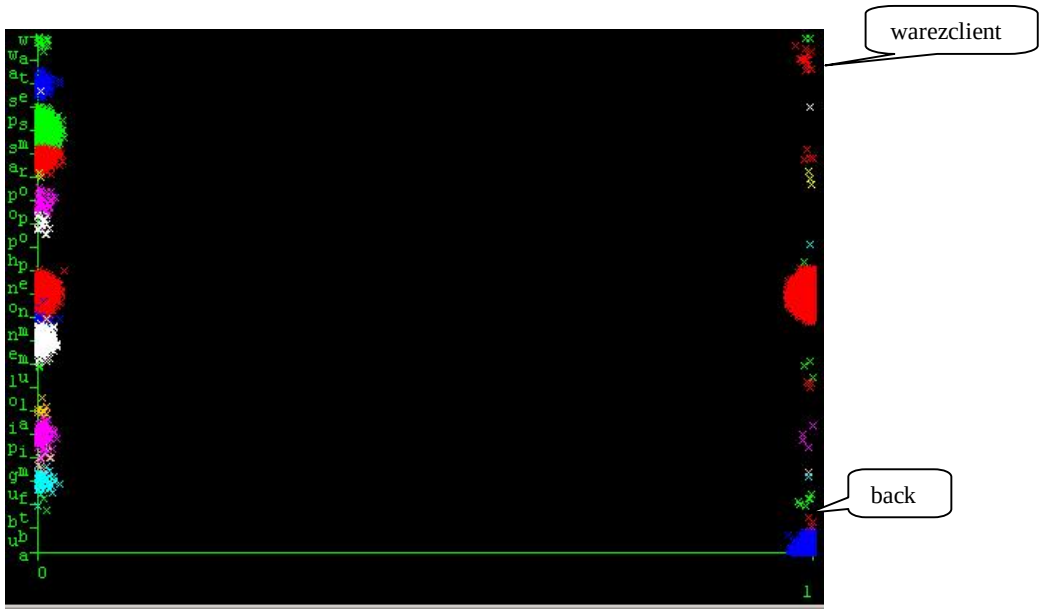


Figura 5.4 Gráfica logged_in - class.

Respecto a los atributos basados en tiempo de "mismo host'", figura 5.5, se puede observar que los ataque de tipo 'smurf' y 'Satán' presentan un número elevado de conexiones a la misma máquina que la conexión actual en los dos últimos segundos.

Se verifica así qué algunos ataques de Probing, como 'Satán', escanean los puertos con un intervalo de tiempo mucho mayor de dos segundos.

Con esto se puede afirmar que los atributos de "mismo host" van a ser de gran utilidad para detectar este tipo de ataques.

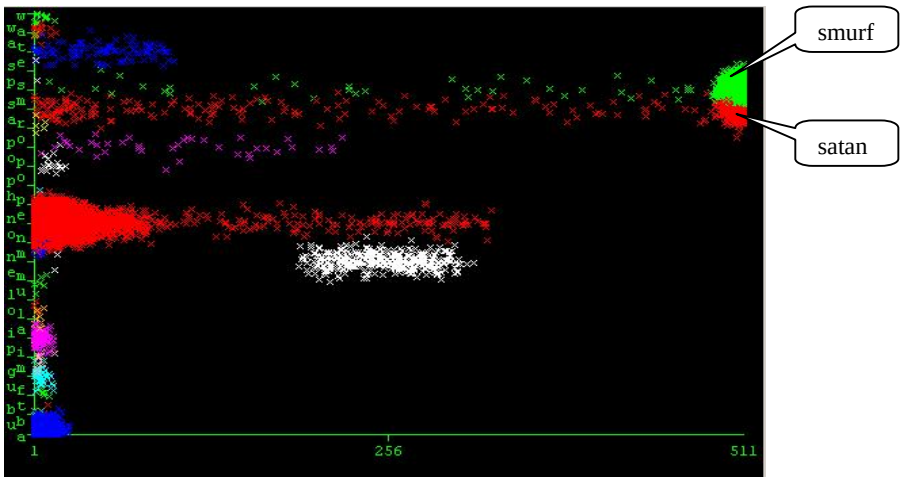


Figura 5.5 Gráfica count - class.

Los atributos basados en tiempo de "mismo servicio", también son útiles para detectar ataques de tipo 'portsweep', 'Satán'

Lo mismo se puede hacer con muchos de los atributos restantes, donde algunos aportan mucha información simplemente mirando las gráficas y con otros no será tan fácil de ver simple vista. Así se intuye cómo puede influir un atributo en la clasificación que hará el modelo de predicción resultante, y decidir si merece la pena su utilización respecto al coste computacional que suponga incluirlo.

5.6 CLASIFICACION DE LOS DATOS

El problema de la clasificación es el más frecuente en la práctica. Para clasificar los datos se ha utilizado la herramienta de minería de datos llamada WEKA, ya mencionada. De esta forma, se van a construir varios modelos que permitan predecir los ataques en función de los atributos de entrada (aprendizaje).

5.6.1 Modos de Evaluación: Clasificador

Para medir la efectividad del algoritmo de clasificación, se va a comparar la clase predicha con la clase real de las instancias. Existen diversos modos de realizar la evaluación:

Use training set: evaluación del clasificador sobre el mismo conjunto sobre el que se construye el modelo predictivo para determinar el error, que en este caso se denominado 'error de resustitución'.

Supplied test set: evalúa sobre un conjunto independiente. Permite cargar un conjunto nuevo de datos. Sobre cada dato se puede realizará una predicción de clase para contar los errores.

Cross-Validation: evaluación con validación cruzada. Se dividen las instancias en tantas carpetas como indica el parámetro 'Folds', y en cada evaluación se toman las instancias de cada carpeta como datos de test, y el resto como datos de entrenamiento para construir el modelo. Los errores calculados serán el promedio de todas las ejecuciones

Percentage split: se dividen los datos en dos grupos, de acuerdo con el porcentaje indicado (%). El valor indicado es el porcentaje de instancias para construir el modelo, que seguidamente es evaluado sobre las que se han dejado aparte.

5.7 Algoritmos basados en Reglas

Algoritmos que aprenden modelos basados en reglas. Mediante los datos de entrenamiento, aprende una serie de reglas y con ellas predice un resultado u otro . Dependiendo del algoritmo que se seleccione, se conseguido mayor o menor eficacia. Se han obtenido los siguientes resultados:

5.7.1 Algoritmo: ZeroR

Este algoritmo es muy simple, asigna a todos los resultados a la clase mayoritaria.

5.7.1.1 Utilizando 42 atributos y validación Cruzada a 10 Folds

Se puede ver que no se obtiene un buen resultado con este clasificador:

En la figura 5.6, el ataque llamado 'normal' se ha producido mas de la mitad de las veces (73.19 % aprox) en los datos de entrenamiento siendo estos la parte mayoritaria. Se observa en la tabla que solo aparece la clase "normal", mientras que las demás instancias no aparecen ni una vez.

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0	0	0	0	0	back
0	0	0	0	0	buffer_overflow
0	0	0	0	0	ftp_write
0	0	0	0	0	guess_passwd
0	0	0	0	0	imap
0	0	0	0	0	ipsweep
0	0	0	0	0	land
0	0	0	0	0	loadmodule
0	0	0	0	0	multihop
0	0	0	0	0	neptune
0	0	0	0	0	nmap
1	1	0.732	1	0.845	normal
0	0	0	0	0	perl
0	0	0	0	0	phf
0	0	0	0	0	pod
0	0	0	0	0	portsweep
0	0	0	0	0	rootkit
0	0	0	0	0	satan
0	0	0	0	0	smurf
0	0	0	0	0	spy
0	0	0	0	0	teardrop
0	0	0	0	0	warezclient
0	0	0	0	0	warezmaster

Figura 5.6. Resultado de clasificación

La siguiente tabla 5.2 muestra la exactitud del modelo respecto al total de las instancias clasificadas

	No. De Instancias	**% de efectividad**
Instancias correctas	17193	73.196 %
Instancias incorrectas	6296	26.804 %

Tabla 5.2 Resultados obtenidos con ZeroR 42 atributos a 10 folds.

La matriz de confusión también nos muestra el número de instancias clasificadazas correctamente ya que como se muestra en la tabla 5.3 y de acuerdo al algoritmo ZeroR solo pudo determinar correctamente el acceso "normal".

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	<-	Clasificado como:
0	0	0	0	0	0	0	0	0	0	0	**975**	0	0	0	0	0	0	0	0	0	0	0		**a= back**
0	0	0	0	0	0	0	0	0	0	0	**5**	0	0	0	0	0	0	0	0	0	0	0		**b= buffer_overflow**
0	0	0	0	0	0	0	0	0	0	0	**6**	0	0	0	0	0	0	0	0	0	0	0		**c= ftp_write**
0	0	0	0	0	0	0	0	0	0	0	**53**	0	0	0	0	0	0	0	0	0	0	0		**d= guess_passwd**
0	0	0	0	0	0	0	0	0	0	0	**12**	0	0	0	0	0	0	0	0	0	0	0		**e= imap**
0	0	0	0	0	0	0	0	0	0	0	**94**	0	0	0	0	0	0	0	0	0	0	0		**f= ipsweep**
0	0	0	0	0	0	0	0	0	0	0	**9**	0	0	0	0	0	0	0	0	0	0	0		**g= land**
0	0	0	0	0	0	0	0	0	0	0	**4**	0	0	0	0	0	0	0	0	0	0	0		**h= loadmodule**
0	0	0	0	0	0	0	0	0	0	0	**6**	0	0	0	0	0	0	0	0	0	0	0		**i= multihop**
0	0	0	0	0	0	0	0	0	0	0	**405**	0	0	0	0	0	0	0	0	0	0	0		**j= neptune**
0	0	0	0	0	0	0	0	0	0	0	**40**	0	0	0	0	0	0	0	0	0	0	0		**k= nmap**
0	0	0	0	0	0	0	0	0	0	0	**17193**	0	0	0	0	0	0	0	0	0	0	0		**l= normal**
0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0		**m= perl**
0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0		**n= phf**
0	0	0	0	0	0	0	0	0	0	0	**20**	0	0	0	0	0	0	0	0	0	0	0		**o= pod**
0	0	0	0	0	0	0	0	0	0	0	**40**	0	0	0	0	0	0	0	0	0	0	0		**p= portsweep**
0	0	0	0	0	0	0	0	0	0	0	**5**	0	0	0	0	0	0	0	0	0	0	0		**q= rootkit**
0	0	0	0	0	0	0	0	0	0	0	**794**	0	0	0	0	0	0	0	0	0	0	0		**r= satan**
0	0	0	0	0	0	0	0	0	0	0	**3691**	0	0	0	0	0	0	0	0	0	0	0		**s= smurf**
0	0	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0	0	0	0		**t= spy**
0	0	0	0	0	0	0	0	0	0	0	**103**	0	0	0	0	0	0	0	0	0	0	0		**u= teardrop**
0	0	0	0	0	0	0	0	0	0	0	**15**	0	0	0	0	0	0	0	0	0	0	0		**v= warezclient**
0	0	0	0	0	0	0	0	0	0	0	**15**	0	0	0	0	0	0	0	0	0	0	0		**w= warezmaster**

Tabla 5.3 Matriz de confusión a 10 folds

Se busca un porcentaje de acierto mayor al 90% para determinar que es posible detectar una intrusión, en este caso solo se consiguió el 73.1% y no cumple con las expectativas.

5.7.1.2 Utilizando 42 atributos y split sample del 66%

El tiempo en construir el modelo fue de 1 seg.

La efectividad de este modelo no a sido la adecuada ya que pudo clasificar correctamente solo el 72.48% la tabla 5.4 muestra las instancias clasificaciones correctas e incorrectas.

	No. De Instancias	**% de efectividad**
Instancias correctas	5786	72.48 %
Instancias incorrectas	2128	27.52 %
Total	7814	100%

Tabla 5.4 Resultados obtenidos con ZeroR 42 atributos y split simple 66%

Al observar la matriz de confusión en la tabla 5.5 nos damos cuenta de que el algoritmo siguió clasificando correctamente la clase mayoritaria “normal” por lo que este algoritmo no es efectivo para predecir las intrusiones.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	<-	
0	0	0	0	0	0	0	0	0	0	0	326	0	0	0	0	0	0	0	0	0	0	0		a= back
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		b= buffer_overflow
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		c= ftp_write
0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0		d= guess_passwd
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		e= imap
0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0		f= ipsweep
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		g= land
0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0		h= loadmodule
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		i= multihop
0	0	0	0	0	0	0	0	0	0	0	135	0	0	0	0	0	0	0	0	0	0	0		j= neptune
0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0		k= nmap
0	0	0	0	0	0	0	0	0	0	0	5789	0	0	0	0	0	0	0	0	0	0	0		l= normal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		m= perl
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		n= phf
0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0		o= pod
0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0		p= portsweep
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		q= rootkit
0	0	0	0	0	0	0	0	0	0	0	281	0	0	0	0	0	0	0	0	0	0	0		r= satan
0	0	0	0	0	0	0	0	0	0	0	1301	0	0	0	0	0	0	0	0	0	0	0		s= smurf
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		t= spy
0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	0	0	0	0	0	0	0		u= teardrop
0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0		v= warezclient
0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0		w= warezmaster

Tabla 5.5 Matriz de confusión con split simple 66%

5.7.1.3 Utilizando 10 atributos y validación Cruzada a 10 Folds

Ahora este modelo se ha probado solo con 10 atributos que han sido los más significativos para esta clasificación, Este modelo se genero en 1 seg. como ya habíamos mencionado este algoritmo basa su clasificación en el atributo mayoritario en este caso es "normal." Los resultados que se muestran en la tabla 5.6 que se han obtenido fueron los mismos que en el caso de la clasificación con 42 atributos, teniendo solo el 73.19% de instancias clasificadas correctamente.

	No. De Instancias	% de efectividad
Instancias correctas	17193	73.196 %
Instancias incorrectas	6296	26.804 %
Total	2398	100%

Tabla 5.6 Resultados obtenidos con 10 atributos a 10 folds.

La matriz de confusión fue exactamente la misma que en la clasificación con 42 atributos y 10 folds considerándose así un algoritmo con muy poca efectividad.

5.7.1.4 Utilizando 10 atributos y split sample del 66%

Este modelo se genero en 1 seg. pero su clasificación no fue la deseada, obtuvimos solamente el 72.48% de aciertos en este algoritmo estos resultados se muestran en la tabla 5.7.

	No. De Instancias	% de efectividad
Instancias correctas	5786	72.48 %
Instancias incorrectas	2128	27.52 %
Total	7814	100%

Tabla 5.7 Resultados obtenidos con 10 atributos y split simple 66%.

Podemos ver que la ventaja de este algoritmo es la rapidez con la que genera y clasifica la información, en este experimento eso no es suficiente ya que se requiere exactitud algo que este algoritmo no da, por consiguiente se descarta completamente este algoritmo.

5.7.1.5 Comparación de resultados ZeroR

En la tabla 5.8 se muestras los resultados obtenidos en los distintos tipos de clasificación donde se puede apreciar que una de las ventajas de estos modelos es el tiempo, el porcentaje de efectividad no ha sido bueno pues como se dijo al principio de este experimento su clasificación se basa a partir de la instancia con mayor numero de apariciones.

	Instancias clasificadas correctamente (%)	Tiempo para construir modelo (s)
Utilizando 42 atributos		
Validación Cruzada a 10 Folds	73.196	1
Split sample del 66%	72.48	1
Utilizando 10 atributos		
Validación Cruzada a 10 Folds	73.196	1
Split sample del 66%	72.48	1

Tabla 5.8 Comparación de resultados obtenidos con ZeroR.

El mayor porcentaje aportado por este modelo es del 73.196 % por lo que no satisface con los resultados que se buscan pues falta casi el 20% para poder cumplirlo.

5.7.2 Algoritmo: Ridor

Este algoritmo genera una regla por defecto que se ajuste a la mayoría de los ejemplos de entrenamiento y luego busca excepciones con la menor tasa de error al clasificar los propios ejemplos de entrenamiento. A continuación genera las excepciones a las excepciones con menos error, de manera recursiva. Así, lleva a cabo una expansión de excepciones en forma de árbol donde la raíz está formada por la regla por defecto. Las excepciones son un conjunto de reglas que predicen las clases que no contempla la regla por defecto.

5.7.2.1 Utilizando 42 atributos y validación Cruzada a 10 Folds

Los resultados que se obtuvieron de este han sido los buscados, ya que el modelo genero 1442 reglas las cuales nos ayudan a determinar que el 99.53% del total de los registros han podido ser clasificados de manera correcta quedando solo 62 instancias mal clasificadas lo que equivale al 0.47 % estas se muestran en la figura 5.7 La tabla 5.9 muestra el tiempo en generar estas reglas, que a sido de 16:09 min.. lo cual implica un inconveniente ya que al tener grandes cantidades de datos esto podría complicar la clasificación debido a que Weka tiene memoria limitada en cuanto al manejo de datos.

```
                Except (logged_in = 1) => ataque = ftp_write  (272.0/0.0) [135.0/0.0]
                        Except (dst_host_count > 2.5) => ataque = warezclient  (9615.0/0.0) [4789.0/0.0]
                                Except (dst_host_srv_count > 7.5) => ataque = normal  (9555.0/0.0) [4775.0/0.0]
                                Except (src_bytes <= 331.5) => ataque = normal  (26.0/0.0) [17.0/0.0]
                                Except (duration <= 5141.5) and (src_bytes > 335.5) => ataque = normal  (14.0/0.0) [5.0/0.0]
                        Except (service = http) => ataque = normal  (309.0/0.0) [148.0/0.0]
                        Except (duration <= 24) and (dst_bytes > 162.5) => ataque = normal  (8.0/0.0) [4.0/0.0]
                        Except (src_bytes > 245.5) and (src_bytes <= 564.5) => ataque = warezclient  (3.0/0.0) [2.0/0.0]
                                Except (dst_host_srv_count > 10) => ataque = normal  (4448.0/0.0) [2225.0/0.0]
                                Except (src_bytes <= 332.5) => ataque = normal  (11.0/0.0) [5.0/0.0]
                                Except (src_bytes > 371.5) => ataque = normal  (4.0/0.0) [2.0/0.0]
                Except (service = domain_u) => ataque = normal  (5.0/0.0) [3.0/0.0]
        Except (hot <= 0.5) => ataque = smurf  (9.0/0.0) [4.0/0.0]
                Except (count <= 328) and (src_bytes <= 1031.5) => ataque = teardrop  (10393.0/0.0) [5184.0/0.0]
                        Except (wrong_fragment <= 0.5) => ataque = warezclient  (10316.0/0.0) [5158.0/0.0]
                                Except (dst_host_srv_count > 4.5) => ataque = normal  (10262.0/0.0) [5135.0/0.0]
                                Except (src_bytes <= 311.5) => ataque = normal  (43.0/0.0) [19.0/0.0]
                                Except (src_bytes > 358.5) => ataque = normal  (5.0/0.0) [2.0/0.0]
                Except (count <= 35.5) and (logged_in = 1) => ataque = warezclient  (264.0/0.0) [133.0/0.0]
                        Except (dst_host_srv_count > 4.5) => ataque = normal  (9654.0/0.0) [4827.0/0.0]
                        Except (dst_host_same_src_port_rate <= 0.8) => ataque = normal  (19.0/0.0) [10.0/0.0]
Except (protocol_type = tcp) => ataque = neptune  (538.0/0.0) [268.0/0.0]
        Except (count <= 148.5) => ataque = guess_passwd  (10267.0/0.0) [5133.0/0.0]
                Except (dst_host_srv_serror_rate <= 0.035) => ataque = warezclient  (10212.0/0.0) [5108.0/0.0]
                        Except (dst_host_srv_count > 7.5) => ataque = ipsweep  (10119.0/0.0) [5057.0/0.0]
                                Except (dst_host_rerror_rate <= 0.825) => ataque = warezmaster  (9905.0/0.0) [4956.0/0.0]
                                        Except (src_bytes > 3) => ataque = normal  (9880.0/0.0) [4936.0/0.0]
                                        Except (duration <= 4.5) => ataque = normal  (26.0/0.0) [13.0/0.0]
                                Except (service = http) => ataque = normal  (199.0/0.0) [94.0/0.0]
                                Except (dst_host_count > 9.5) => ataque = warezmaster  (12.0/0.0) [6.0/0.0]
                                        Except (dst_host_same_src_port_rate <= 0.585) => ataque = normal  (8878.0/0.0) [4435.0/0.0]
                                        Except (duration <= 4.5) => ataque = normal  (26.0/0.0) [12.0/0.0]
                        Except (src_bytes <= 332.5) => ataque = ftp_write  (71.0/0.0) [36.0/0.0]
```

Figura 5.7 Reglas obtenidas con validación cruzada con 42 atributos a 10 folds.

	No. De Instancias	% de efectividad
Instancias correctas	23378	99.53 %
Instancias incorrectas	111	0.47 %
Total	23489	100%

Tabla 5.9 Resultados obtenidos con 42 atributos a 10 folds.

La matriz de confusión muestra la efectividad de este método ya que se puede apreciar la buena clasificación que realizo tabla 5.10.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	<-	
974	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		**a= back**
0	4	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		**b= buffer_overflow**
0	0	1	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0		**c= ftp_write**
0	0	0	51	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		**d= guess_passwd**
0	0	0	0	10	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		**e= imap**
0	0	0	0	0	94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		**f= ipsweep**
0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		**g= land**

0	1	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**h= loadmodule**
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	1	2		**i= multihop**
0	0	0	0	0	0	0	0	0	405	0	0	0	0	0	0	0	0	0	0	0	0	0		**j= neptune**
0	0	0	0	0	0	0	0	1	0	40	0	0	0	0	0	0	0	0	0	0	0	0		**k= nmap**
1	0	0	0	0	0	1	0	1	1	0	17176	2	0	0	2	2	1	3	0	0	0	3		**l= normal**
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		**m= perl**
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		**n= phf**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0		**o= pod**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37	0	3	0	0	0	0	0		**p= portsweep**
0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	2	0	0	0	0	0	0		**q= rootkit**
0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	790	0	0	0	0	0		**r= satan**
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	3689	0	0	0		**s= smurf**
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		**t= spy**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	0	0		**u= teardrop**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0		**v= warezclient**
0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	1	11		**w= warezmaster**

Tabla 5.10 Matriz de confusión con 10 folds.

5.7.2.2 Utilizando 42 atributos y split sample del 66%

Este modelo fue generado en 11.52 min. y creo las mismas 1442 reglas que usamos en la validación a 10 folds. Vemos en la tabla 5.11 que a pensar de solo tomar parte de las instancias el algoritmo sigue siendo muy convincente ya que solo clasifico erróneamente el 0.40% mientras que el 99.59 % a sido clasificado correctamente. A pesar de ser muy efectivo este método el inconveniente sigue siendo el tiempo.

	No. De Instancias	% de efectividad
Instancias correctas	7939	99.59 %
Instancias incorrectas	48	0.40 %
Total	7987	100%

Tabla 5.11 Resultados obtenidos con 42 atributos y split simple 66%.

La matriz de confusión muestra prácticamente el mismo comportamiento que en la validación a 10 folds mostrando la clasificación satisfactoria del ataque “normal”.

5.7.2.3 Utilizando 10 atributos y validación Cruzada a 10 Folds

El algoritmo Ridor genero el modelo de clasificación en 2.39 min. generando 946 reglas las cuales obtienen un 99.42 % de efectividad en la clasificación de los ataques. La tabla 5.12 muestra de manera más detallada este hecho.

	No. De Instancias	% de efectividad
Instancias correctas	23361	99.42 %
Instancias incorrectas	137	0.58 %
Total	23498	100%

Tabla 5.12 Resultados obtenidos con 10 atributos a 10 folds.

Las 946 reglas generadas figura 5.8 han mostrado que es efectivo el algoritmo al clasificar, pero a pesar de ser bueno tardo 38 min en clasificar las 23498 instancias, esto como ya se dijo podría ocasionar problemas de memoria si no se cuenta con un equipo con suficientes recurso.

```
                        Except (service = http) and (dst_host_count > 61.5) => ataque = normal  (0.0/0.0) [1.0/0.0]
                        Except (service = other) => ataque = normal  (8.0/0.0) [2.0/0.0]
                                Except (count > 314) => ataque = satan  (385.0/0.0) [194.0/0.0]
                                Except (count > 3) => ataque = satan  (14.0/0.0) [6.0/0.0]
                        Except (service = time) => ataque = normal  (2.0/1.0) [2.0/1.0]
                Except (dst_bytes <= 2584589.5) and (dst_bytes > 219.5) => ataque = normal  (22.0/0.0) [9.0/0.0]
                Except (service = finger) => ataque = normal  (3.0/0.0) [1.0/0.0]
        Except (protocol_type = udp) => ataque = normal  (6.0/0.0) [3.0/0.0]
                Except (src_bytes <= 9) => ataque = satan  (50.0/0.0) [24.0/0.0]
 Except (dst_host_count <= 247.5) => ataque = portsweep  (26.0/0.0) [13.0/0.0]
        Except (dst_host_count <= 177.5) => ataque = satan  (664.0/0.0) [327.0/0.0]
                Except (count <= 5.5) => ataque = warezmaster  (600.0/0.0) [305.0/0.0]
                        Except (dst_bytes <= 168) => ataque = ipsweep  (579.0/0.0) [287.0/0.0]
                                Except (src_bytes > 2.5) => ataque = normal  (340.0/0.0) [171.0/0.0]
                                Except (service = http) and (dst_host_count <= 59.5) => ataque = normal  (150.0/0.0) [65.0/0.0]
                                Except (service = http) and (dst_host_count > 62) => ataque = normal  (16.0/0.0) [11.0/0.0]
                                Except (service = other) => ataque = normal  (7.0/0.0) [3.0/0.0]
                                Except (service = imap4) => ataque = imap  (3.0/0.0) [3.0/0.0]
                                Except (service = http) => ataque = normal  (5.0/2.0) [2.0/1.0]
                                Except (service = time) => ataque = normal  (2.0/1.0) [2.0/1.0]
                        Except (src_bytes > 2) => ataque = normal  (20.0/0.0) [8.0/0.0]
                Except (service = domain_u) => ataque = normal  (44.0/0.0) [16.0/0.0]
                Except (service = ftp_data) => ataque = normal  (8.0/0.0) [2.0/0.0]
                        Except (dst_bytes > 2575386) => ataque = warezmaster  (7.0/0.0) [3.0/0.0]
                Except (service = http) => ataque = ipsweep  (3.0/0.0) [1.0/0.0]
                        Except (dst_host_count <= 60.5) => ataque = normal  (145.0/0.0) [75.0/0.0]
                        Except (dst_host_count > 61.5) => ataque = normal  (20.0/0.0) [9.0/0.0]
        Except (src_bytes > 3.5) => ataque = normal  (64.0/0.0) [33.0/0.0]
                Except (count > 6.5) and (protocol_type = tcp) and (src_bytes <= 56.5) => ataque = satan  (3.0/0.0) [1.0/0.0]
        Except (count > 68.5) => ataque = normal  (3.0/0.0) [2.0/0.0]
                Except (protocol_type = tcp) => ataque = satan  (4.0/0.0) [2.0/0.0]
 Except (src_bytes > 95.5) => ataque = normal  (2.0/0.0) [1.0/0.0]
count > 79) => ataque = pod  (3.0/0.0) [1.0/0.0]
```

Figura 5.8 Reglas obtenidas con validación cruzada con 10 atributos a 10 folds.

5.7.2.4 Utilizando 10 atributos y split sample del 66%

Modelo generado en 2:25 min. El tiempo en comparación a otros modelos sigue siendo mayor, por lo que se considerara en usar o no este modelo, pese a que cumple con los resultados buscados, estos resultados se muestran en la tabla 5.13. Generando 946 reglas con una efectividad del 99.3 %.

	No. De Instancias	% de efectividad
Instancias correctas	7931	99.3 %
Instancias incorrectas	56	0.7 %
Total	7987	100%

Tabla 5.13 Resultados obtenidos con 10 atributos y split simple 66%.

Claramente se puede apreciar en la matriz de confusión tabla 5.14 estos resultados, pues se logra ver que la mayor parte de los errores pertenecen a la instancia "normal" pues es la que predomina en el total de las instancias.

```
  a   b  c   d  e   f  g  h  i   j   k    l   m  n  o  p  q   r    s   t  u  v  w  <-
 325  0  0   0  0   0  0  0  0   0   0    1   0  0  0  0  0   0    0   0  0  0  0  0      a= back
   0  1  0   0  0   0  0  0  0   0   0    1   0  0  0  0  0   0    0   0  0  0  0  0      b= buffer_overflow
   0  0  0   1  0   0  0  0  0   0   0    1   0  0  0  0  0   0    0   0  0  0  0  0      c= ftp_write
   0  0  0  19  0   0  0  0  0   0   0    0   0  0  0  0  0   0    0   0  0  0  0  0      d= guess_passwd
```

0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**e= imap**
0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	**f= ipsweep**
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**g= land**
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	**h= loadmodule**
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	**i= multihop**
0	0	0	0	0	0	0	0	0	135	0	0	0	0	0	0	0	0	0	0	0	0	0	**j= neptune**
0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	**k= nmap**
0	0	0	0	0	7	0	0	0	0	0	5780	0	0	0	0	0	1	0	1	0	0	0	**l= normal**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**m= perl**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**n= phf**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	**o= pod**
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	1	0	0	0	0	0	**p= portsweep**
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	**q= rootkit**
0	0	0	0	0	0	0	0	0	9	0	1	0	0	0	0	0	271	0	0	0	0	0	**r= satan**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1301	0	0	0	0	**s= smurf**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**t= spy**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	**u= teardrop**
0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	2	0	**v= warezclient**
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	4	**w= warezmaster**

Tabla 5.14 Matriz de confusión con split simple 66%.

5.7.2.5 Comparación de resultados Ridor

Los resultados obtenidos en este modelo han sido muy buenos estos se muestran en la tabla 5.15, ya que sobrepasa el porcentaje de aciertos buscados, con un 99.53% se presenta como el mejor en cuanto a clasificación, pero en cuestión de tiempo ha sido el peor, pues la clasificación a 10 folds la realizo en 16:09 min, lo que es un inconveniente ya que si se desean clasificar mas de las 22000 instancias el tiempo superaría ese tiempo, además solo se tomo un pequeño porcentaje de la gran cantidad de instancias que día a día se guardan el el archivo de datos.

	Instancias clasificadas correctamente (%)	**Tiempo para construir modelo (m)**
Utilizando 42 atributos		
Validación Cruzada a 10 Folds	99.53	16:09
Split sample del 66%	99.40	11:52
Utilizando 10 atributos		
Validación Cruzada a 10 Folds	99.42	2.39
Split sample del 66%	99.3	2.25

Tabla 5.15 Comparacion de resultados obtenidos Ridor.

Aunque es cierto que la clasificación con 10 atributos mejoro por mucho el tiempo aun sigue siendo mayor a comparación de los demás modelos, sin lugar a dudas estos resultados estarían dentro de los 3 mejores clasificadores, pero lo descartamos como el mejor.

5.8 Algoritmo: Naive Bayes

Se ha hecho mención este tipo de algoritmos porque alguna de sus variantes tiene la característica de que se ve gravemente afectada por atributos irrelevantes a la hora de realizar el aprendizaje.

5.8.1 Utilizando 42 atributos y validación Cruzada a 10 Folds

Este modelo se genero en 1.49 seg. obteniendo el 90.04 % de instancias clasificadas correctamente mientras que solo el 9.95% de instancias clasificadas incorrectamente. Véase tabla 5.16.

	No. De Instancias	% de efectividad
Instancias correctas	21149	90.04 %
Instancias incorrectas	2340	9.95 %
Total	23489	100%

Tabla 5.16 Resultados obtenidos con 42 atributos a 10 folds.

Cabe mencionar que aunque se encontró el porcentaje buscado en la clasificación, el error absoluto figura 5.9 es mayor que en los anteriores modelos, tendiendo este el 11.36% siendo muy alto para la clasificación.

```
Kappa statistic                          0.8938
Mean absolute error                      0.0043
Root mean squared error                  0.0642
Relative absolute error                 11.3626 %
Root relative squared error             46.6108 %
Total Number of Instances           23489
```

Figura 5.9 Error absoluto de la validación con 10 folds..

La matriz de confusión generada muestra en la tabla 5.17 que existen muchos errores al clasificar el tipo de ataque “normal”, lo que genera que no sea confiable pues muchas de las instancias serian clasificadas así.

```
  a   b   c   d   e   f   g   h   i   j   k    l   m   n   o   p   q   r    s   t   u   v   w  <-
318   0   0   0   0   0   0   0   0   0   0    8   0   0   0   0   0   0    0   0   0   0   0      a= back
  0   2   0   0   0   0   0   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      b= buffer_overflow
  0   0   2   0   0   0   0   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      c= ftp_write
  0   0   0  19   0   0   0   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      d= guess_passwd
  0   0   0   0   1   0   0   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      e= imap
  0   0   0   0   0  34   0   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      f= ipsweep
  0   0   0   0   0   0   2   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      g= land
  0   1   0   0   0   0   0   0   0   0   0    1   0   0   0   0   0   0    0   0   0   0   0      h= loadmodule
  0   0   0   0   0   0   0   0   1   0   0    0   0   0   0   0   0   0    0   0   0   0   0      i= multihop
  0   0   0   0   0   0   0   0   0 135   0    0   0   0   0   0   0   0    0   0   0   0   0      j= neptune
  0   0   0   0   0   0   0   0   4   0  15    0   0   0   0   0   0   0    0   0   0   0   0      k= nmap
 53   3  89  36   4  16   0   0   0   0   0 5443   0   0   1   1   3  34    4   0  16  27  55      l= normal
  0   0   0   0   0   0   0   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      m= perl
  0   0   0   0   0   0   0   0   0   0   0    0   0   0   0   0   0   0    0   0   0   0   0      n= phf
  0   0   0   0   0   0   0   0   0   0   0    0   0   0   6   0   0   0    0   0   0   0   0      o= pod
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	1	0	0	0	0	0	p= portsweep
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	q= rootkit
0	0	0	0	0	0	0	0	0	0	0	10	0	0	1	5	0	265	0	0	0	0	0	r= satan
0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	6	1291	0	0	0	0	s= smurf
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	t= spy
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	u= teardrop
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	v= warezclient
0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	1	4	w= warezmaster

Tabla 5.17 Matriz de confusión con 10 folds.

Una ventaja de este modelo es el tiempo en realizar la clasificación, este es uno que podríamos considerar para nuestro trabajo.

5.8.2 Utilizando 42 atributos y split sample del 66%

Este modelo se genero en 1.38 seg. reafirmando la eficacia de este algoritmo en cuanto al tiempo, este modelo nos da buenos resultados pues clasifico correctamente el 89.15% las instancias que se muestran en la tabla 5.18.

	No. De Instancias	% de efectividad
Instancias correctas	7120	89.15 %
Instancias incorrectas	867	10.84 %
Total	7987	100%

Tabla 5.18 Resultados obtenidos con 42 atributos con split simple 66%.

Aunque podríamos considerar como bueno el algoritmo, no ha cumplido con lo buscado, a pesar de su velocidad al clasificar existen algoritmos con mejores resultados y con menor margen de error, por lo tanto se descarta este modelo para la clasificación de los ataques.

5.8.3 Utilizando 10 atributos y validación Cruzada a 10 Folds

Modelo generado en 38 seg. muy eficaz en cuestión de tiempo, pero solo nos brindo un 83.69 % de efectividad en esta clasificación. y dando 3832 instancias clasificadas de manera incorrecta. La tabla 5.19 muestra con claridad estos resultados.

	No. De Instancias	% de efectividad
Instancias correctas	19666	83.69 %
Instancias incorrectas	3832	16.31 %
Total	23498	100%

Tabla 5.19 Resultados obtenidos con 10 atributos a 10 folds.

La matriz de confusión 5.20 muestra de manera mas detallada estos errores en la clasificación. La mayoría de estos están dentro de la clase “normal”. Esto debido a la gran cantidad de registros que existen con esta clase.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	<-
932	0	0	0	0	0	0	0	0	0	0	43	0	0	0	0	0	0	0	0	0	0	0	a= back
0	4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	b= buffer_overflow

0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	2	0	c= ftp_write
0	0	0	52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	d= guess_passwd
0	0	0	2	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	e= imap
0	0	0	4	0	89	0	1	0	0	0	0	0	0	0	2	1	0	0	0	0	0	f= ipsweep
0	0	0	0	0	2	6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	g= land
0	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	h= loadmodule
0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	i= multihop
0	0	0	0	0	0	0	0	0	403	0	0	0	0	0	0	0	2	0	0	0	0	j= neptune
0	0	0	2	0	0	0	0	0	0	38	0	0	0	0	0	0	0	0	0	0	0	k= nmap
229	131	59	138	4	957	144	121	17	55	11	13589	0	4	5	30	490	247	0	112	1	839	10	l= normal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	m= perl
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	n= phf
0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	o= pod
0	0	0	0	0	0	0	0	0	14	0	1	0	0	0	3	0	22	0	0	0	0	p= portsweep
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	4	0	0	0	0	0	q= rootkit
0	0	0	0	0	0	0	0	0	41	0	9	0	0	2	4	2	735	0	0	0	1	r= satan
0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	16	3661	0	0	0	s= smurf
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	t= spy
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	101	0	u= teardrop
0	0	0	0	0	2	0	4	0	0	0	0	0	0	0	0	2	0	0	5	0	2	v= warezclient
0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	11	w= warezmaster

Tabla 5.20 Matriz de confusión con 10 folds.

5.8.4 Utilizando 10 atributos y split sample del 66%

El tiempo estimado de la construcción del modelo fue de 28 seg. con una efectividad del 86.07% desafortunadamente los resultados no fueron los requeridos aunque volvemos a hacer hincapié en el tiempo de respuesta de este algoritmo. La tabla 5.21 muestra los resultados finales.

	No. De Instancias	% de efectividad
Instancias correctas	6875	86.07 %
Instancias incorrectas	1112	13.93 %
Total	7987	100%

Tabla 5.21 Resultados obtenidos con 10 atributos y split simple 66%.

5.8.5 Comparación de resultados Naive Bayes

Claramente se puede apreciar en la tabla 5.22 que este modelo no cumple con el 90% buscado para la clasificación, los tiempos y demás resultados son buenos, pues el mejor tiempo que obtuvo dieron 28 seg. en la clasificación con 10 atributos y utilizando split sample al 66%. Debido a que se obtuvieron mejores resultados con otros modelos este se descartara.

	Instancias clasificadas correctamente (%)	Tiempo para construir modelo (s)
Utilizando 42 atributos		
Validación Cruzada a 10 Folds	90.04	109
Split sample del 66%	89.15	98
Utilizando 10 atributos		
Validación Cruzada a 10 Folds	83.69	38
Split sample del 66%	86.07	28

Tabla 5.22 Comparacion de resultados obtenidos con Naive Bayes.

5.9 Algoritmo: J48

El algoritmo J48 de WEKA es una implementación del algoritmo C4.5, uno de los algoritmos de minería de datos más utilizado. Se trata de un refinamiento del modelo generado con OneR el cual selecciona el atributo que mejor explica la clase de salida

5.9.1 Utilizando 42 atributos y validación Cruzada a 10 Folds

Modelo generado en 22.83 seg. tomando como nodo raíz "srv_count" , y obtiene un buen resultado, este árbol mostrado en la figura 5.10 que genero 395 reglas tuvo una efectividad del 97.11% mientras que solo 1650 instancias fueron clasificadas de manera incorrecta..

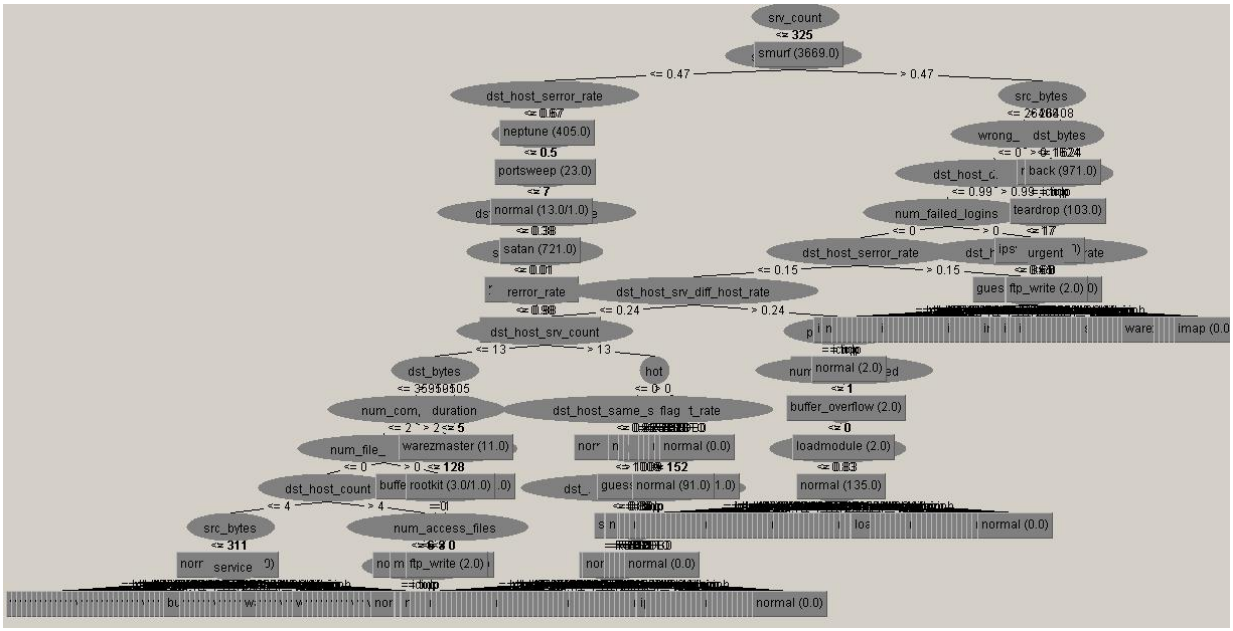


Figura 5.10. Árbol generado con J48 y 42 atributos.

La tabla 5.23 muestra los resultados obtenidos en la clasificación:

	No. De Instancias	% de efectividad
Instancias correctas	21839	97.11 %
Instancias incorrectas	1650	2.89 %
Total	23489	100%

Tabla 5.23 Resultados obtenidos con 42 atributos a 10 folds.

Supone una mejora a considerar respectos a los demás algoritmos, tiene una efectividad del 97.11% superior a lo que se busca. Esperaremos comos e comporta en las demás pruebas.

5.9.2 Utilizando 42 atributos y split sample del 66%

Este modelo fue generado en 16.74 seg. con una clasificación de instancias correctas del 97.05% como vemos en la tabla 5.24 son muy parecidos los resultados a los obtenidos en el caso anterior. De la misma manera se generaron: 395 reglas en este árbol. Y el grafico se generado es idéntico al anterior.

	No. De Instancias	% de efectividad
Instancias correctas	7751	97.05 %
Instancias incorrectas	236	2.95 %
Total	7987	100%

Tabla 5.24 Resultados obtenidos con 42 atributos con split simple 66%.

Es interesante notar que el error absoluto de este modelo mostrado en la figura 5.11 ha sido menor que en el de casos anteriores lo cual es un buen indicativo de que la clasificación sea correcta.

```
Kappa statistic                          0.9738
Mean absolute error                      0.0003
Root mean squared error                  0.0151
Relative absolute error                  0.8465 %
Root relative squared error             10.8528 %
Total Number of Instances             7987
```

Figura 5.11. Error absoluto con split simple 66%.

La tabla 5.25 muestra en la matriz de confusión los errores de esta clasificación, podemos ver q son poco los q se presentan y la mayor parte de ellos es en la clase normal.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	<-- classified as
324	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	a = back
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	b = buffer_overflow
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	c = ftp_write
0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	d = guess_passwd
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	e = imap
0	0	0	0	0	34	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	f = ipsweep
0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	g = land
0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	h = loadmodule
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	i = multihop
0	0	0	0	0	0	0	0	0	135	0	0	0	0	0	0	0	0	0	0	0	0	0	j = neptune
0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	k = nmap
0	0	0	0	0	0	0	0	0	0	0	5785	0	0	0	0	0	0	0	1	0	1	2	l = normal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	m = perl
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	n = phf
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	o = pod
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	2	0	0	0	0	0	p = portsweep
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	q = rootkit
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	279	0	0	0	0	0	r = satan
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1299	0	0	0	0	s = smurf
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	t = spy
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	u = teardrop
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	v = warezclient
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	w = warezmaster

Tabla 5.25 Matriz de confusión con split simple 66%..

5.9.3 Utilizando 10 atributos y validación Cruzada a 10 Folds

El tiempo estimado de la construcción del modelo fue de 5.3 seg. Tomemos en cuenta la eficiencia de este modelo pues genero un 99.3 % de instancias clasificadas correctamente. Ver tabla 5.26.

	No. De Instancias	% de efectividad
Instancias correctas	23324	99.3 %
Instancias incorrectas	164	0.69 %
Total	23489	100%

Tabla 5.26 Resultados obtenidos con 10 atributos a 10 folds.

El árbol generado esta construido por 391 reglas que concluyen en las hojas para poder evaluar el modelo. La figura 5.12 muestra de manera grafica este árbol.

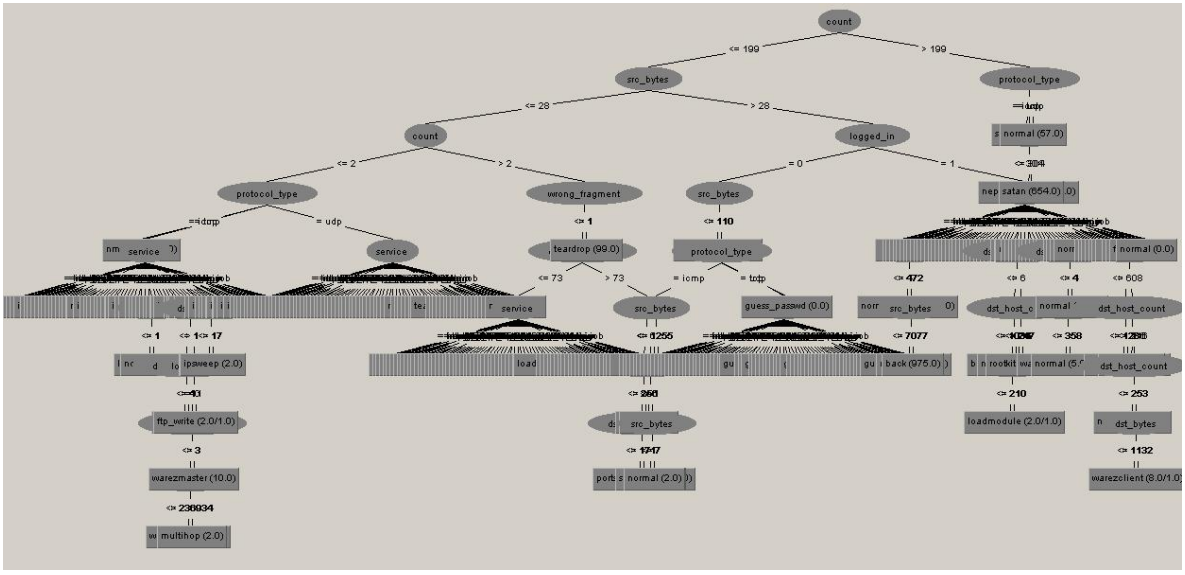


Figura 5.12. Árbol construido con 10 atributos y 10 folds.

5.9.4 Utilizando 10 atributos y split sample del 66%

Los resultados obtenidos al evaluar las 7987 instancias de prueba muestran se de forma general en la tabla 5.27, en la que se observa que el modelo tiene una efectividad del 99.27% y un error de 0.73%. El tiempo de construcción es de los mejores pues solo tardo 2.1 seg.

	No. De Instancias	% de efectividad
Instancias correctas	7928	99.27 %
Instancias incorrectas	59	0.73 %
Total	7987	100%

Tabla 5.27 Resultados obtenidos con 10 atributos y split simple 66%.

De forma más detallada se presentan los resultados en la matriz de confusión en la tabla 5.28 que se puede apreciar que la clase “normal” presenta más errores.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	<-	
325	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		a= back
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		b= buffer_overflow
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		c= ftp_write
0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		d= guess_passwd
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		e= imap
0	0	0	0	0	30	1	0	0	0	0	4	0	0	0	2	1	0	0	0	0	0	0		f= ipsweep
0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		g= land
0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		h= loadmodule
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0		i= multihop
0	0	0	0	0	0	0	0	0	135	0	0	0	0	0	0	0	2	0	0	0	0	0		j= neptune
0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0		k= nmap
0	0	0	0	0	2	0	0	0	0	0	5787	0	0	0	0	0	0	0	0	0	0	0		l= normal

```
0  0  0   0  0   0  0  0  0   0   0    0  0  0  0  0  0    0    0  0   0  1  0    m= perl
0  0  0   0  0   0  0  0  0   0   0    0  0  0  0  0  0    0    0  0   0  0  0    n= phf
0  0  0   0  0   0  0  0  0   0   0    0  0  0  6  0  0    0    0  0   0  0  0    o= pod
0  0  0   0  0   0  0  0  0   1   0    0  0  0  0  0  0   13    0  0   0  0  0    p= portsweep
0  0  0   0  0   0  0  0  0   0   0    2  0  0  0  0  0    0    0  0   0  0  0    q= rootkit
0  0  0   0  0   0  0  0  0   9   0    1  0  0  0  0  0  271    0  0   0  1  0    r= satan
0  0  0   0  0   0  0  0  0   0   0    0  0  0  0  0  0    0 1301  0   0  0  0    s= smurf
0  0  0   0  0   0  0  0  0   0   0    0  0  0  0  0  0    0    0  0   0  0  0    t= spy
0  0  0   0  0   0  0  0  0   0   0    0  0  0  0  0  0    0    0  0  42  0  0    u= teardrop
0  0  0   0  0   0  0  0  0   0   0    6  0  0  0  0  0    0    0  0   0  0  0    v= warezclient
0  0  0   0  0   0  0  1  1   0   0    1  0  0  0  0  0    0    0  0   0  0  2    w= warezmaster
```

Tabla 5.28 Matriz de confusión con 10 atributos y split simple 66%.

El árbol generado fue exactamente igual al de la validación a 10 folds generando las mismas 391 reglas que se muestran en la figura 5.13.

```
count <= 199
|   src_bytes <= 28
|   |   count <= 2
|   |   |   protocol_type = icmp: nmap (41.0/1.0)
|   |   |   protocol_type = tcp
|   |   |   |   service = imap4: imap (5.0)
|   |   |   |   service = auth: normal (36.0)
|   |   |   |   service = http: normal (415.0/3.0)
|   |   |   |   service = efs: normal (0.0)
|   |   |   |   service = http_2784: normal (0.0)
|   |   |   |   service = red_i: normal (0.0)
|   |   |   |   service = nnsp: normal (0.0)
|   |   |   |   service = rje: ipsweep (2.0)
|   |   |   |   service = sql_net: normal (0.0)
|   |   |   |   service = supdup: normal (0.0)
|   |   |   |   service = bgp: normal (0.0)
|   |   |   |   service = daytime: normal (0.0)
|   |   |   |   service = netstat: normal (0.0)
|   |   |   |   service = http_443: normal (0.0)
|   |   |   |   service = aol: normal (0.0)
```

Figura 5.13. Reglas generadas con 10 atributos y split simple 66%.

5.9.5 Comparación de resultados J48

Los resultados de este modelo se presentan el la tabla 5.29 donde se puede observar variaciones a tomar en cuanta pues los resultados difieren en buena medida al numero de atributos seleccionados. Mientas que con 42 clasifica el 97% con solo 10 obtiene el 99% , dando buenos resultados pues el tiempo es un favor muy importante a considerar y solo se tardo 5.3 seg. en clasificar las mas de 22000 instancias.

	Instancias clasificadas correctamente (%)	**Tiempo para construir modelo (s)**
Utilizando 42 atributos		
Validación Cruzada a 10 Folds	97.11	22.83
Split sample del 66%	97.05	16.74

Utilizando 10 atributos		
Validación Cruzada a 10 Folds	99.3	5.3
Split sample del 66%	99.27	2.1

Tabla 5.29 Comparación de resultados obtenidos J48.

En este caso los tributos de mayor peso juegan un papel fundamental pues solo con 10 de ellos obtenemos una mejor clasificación que si usaremos los 42, podemos decir que este es un buen modelo, para determinar de manera clara una buena clasificación de intrusión.

5.10 Algoritmo: Random Forest

Un algoritmo clasificador que consiste en una colección de L clasificadores con estructura de árbol, donde cada árbol se ha construido a partir de un vector aleatorio.

5.10.1 Utilizando 42 atributos y validación Cruzada a 10 Folds

La construcción de este modelo se genero en 10.6 seg. proporcionando muy buenos resultados en la tabla 5.30 muestra la efectividad del mismo.

	No. De Instancias	**% de efectividad**
Instancias correctas	23286	98..71 %
Instancias incorrectas	303	1.29 %
Total	23489	100%

Tabla 5.30 Resultados obtenidos con 42 atributos a 10 folds.

La matriz de confusión en la tabla 5.31 muestra claramente la efectividad de este modelo ya que se puede apreciar que los errores son pocos en relación a la cantidad total de instancias.

```
  a   b   c   d   e   f   g   h   i   j   k     l   m   n   o   p   q   r   s   t   u   v   w   <-
975   0   0   0   0   0   0   0   0   0   0     0   0   0   0   0   0   0   0   0   0   0   0      a= back
  0   5   0   0   0   0   0   0   0   0   0     0   0   0   0   0   0   0   0   0   0   0   0      b= buffer_overflow
  0   0   3   0   0   0   0   0   0   0   0     3   0   0   0   0   0   0   0   0   0   0   0      c= ftp_write
  0   0   0  51   0   0   0   0   0   0   0     2   0   0   0   0   0   0   0   0   0   0   0      d= guess_passwd
  0   0   0   0  11   0   0   0   0   0   0     1   0   0   0   0   0   0   0   0   0   0   0      e= imap
  0   0   0   0   0  86   0   0   0   0   0     6   0   0   0   2   0   0   0   0   0   0   0      f= ipsweep
  0   0   0   0   0   0   9   0   0   0   0     0   0   0   0   0   0   0   0   0   0   0   0      g= land
  0   1   0   0   0   0   0   3   0   0   0     0   0   0   0   0   0   0   0   0   0   0   0      h= loadmodule
  0   1   0   0   0   0   0   0   4   0   0     0   0   0   0   0   0   0   0   0   0   1   0      i= multihop
  0   0   0   0   0   0   0   0   0 405   0     0   0   0   0   0   0   0   0   0   0   0   0      j= neptune
  0   0   0   0   0   0   0   0   0   0  39     1   0   0   0   0   0   0   0   0   0   0   0      k= nmap
  0   0   1   1   0   1   0   0   0   1   0 17090   0   0   0   0   0   1   3   0   0   0   3      l= normal
  0   0   0   0   0   0   0   0   0   0   0     1   0   0   0   0   0   0   0   0   0   0   0      m= perl
  0   0   0   0   0   0   0   0   0   0   0     1   0   0   0   0   0   0   0   0   0   0   0      n= phf
  0   0   0   0   0   0   0   0   0   0   0     0   0   0  20   0   0   0   0   0   0   0   0      o= pod
  0   0   0   1   0   0   0   0   0   0   0     1   0   0   0  37   0   1   0   0   0   0   0      p= portsweep
```

```
0  1  0   0   0   0  0  0  1   0   0     1  0  0   0   0  2   0     0  0    0   0   0      q= rootkit
0  0  0   0   0   0  0  0  0   0   0     7  0  0   0   0  0 787     0  0    0   0   0      r= satan
0  0  0   0   0   0  0  0  0   0   0     2  0  0   0   0  0   0  3689  0    0   0   0      s= smurf
0  0  0   0   0   0  0  0  0   0   0     2  0  0   0   0  0   0     0  0    0   0   0      t= spy
0  0  0   0   0   0  0  0  0   0   0     0  0  0   0   0  0   0     0  0  103   0   0      u= teardrop
0  0  0   0   0   0  0  0  0   0   0     0  0  0   0   0  0   0     0  0    0  15   0      v= warezclient
0  0  0   0   0   0  0  0  4   0   0     1  0  0   0   0  0   0     0  0    0   1  10      w= warezmaster
```

Tabla 5.31 Matriz de confusión con 42 atributos a 10 folds.

Podemos decir que este algoritmo es candidato para utilizar en este experimento por los buenos resultados alcanzados.

5.10.2 Utilizando 42 atributos y split sample del 66%

Este modelo se construyo en solo 9.57 seg. como se puede apreciar no es mucha la diferencia en comparación con el modelo utilizado anteriormente, en este caso los resultados también han sido favorables pues mostró 98.54% de efectividad y solo el 1.46% de error. Los resultados se aprecian en la tabla 5.32.

	No. De Instancias	**% de efectividad**
Instancias correctas	7870	98.54 %
Instancias incorrectas	117	1.46 %
Total	7987	100%

Tabla 5.32 Resultados obtenidos con 42 atributos y split simple 66%.

La matriz de confusión muestra mas errores en el ataque “normal” ya que las instancias con mayor aparición es la de este tipo, aunque en total son 117 errores podemos decir que el modelo satisface con los resultados a buscar.

5.10.3 Utilizando 10 atributos y validación Cruzada a 10 Folds

Algoritmo muy eficaz pues pudo generar un 99.3% de efectividad en solo 5.06 seg. superando las expectativas planteada al principio, la tabla 5.33 muestra a detalle estos resultados.

	No. De Instancias	**% de efectividad**
Instancias correctas	23324	99.3 %
Instancias incorrectas	164	0.69 %
Total	23489	100%

Tabla 5.33 Resultados obtenidos con 10 atributos a 10 folds.

En la matriz de confusión mostrada en la tabla 5.34 apreciamos los pocos errores que tuvo este modelo pese a clasificar mas de 23000 instancias.

Vemos pues que los resultados fueron muy alentadores pues cumple con en mínimo de efectividad que es el 90% y además es un algoritmo que clasifica muy rápido, considerándolo para la evaluación final.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	<-	
975	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		a= back
0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		b= buffer_overflow
0	0	3	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0		c= ftp_write
0	0	0	52	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		d= guess_passwd
0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		e= imap
0	0	0	1	0	81	1	0	0	0	0	9	0	0	0	2	0	0	0	0	0	0	0		f= ipsweep
0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		g= land
0	0	1	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		h= loadmodule
0	0	0	0	0	1	0	0	1	0	0	2	0	0	0	0	2	0	0	0	0	0	1		i= multihop
0	0	0	0	0	0	0	0	0	403	0	0	0	0	0	0	0	2	0	0	0	0	0		j= neptune
0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0		k= nmap
0	0	1	2	0	5	0	0	0	0	0	17180	1	0	0	0	0	0	0	0	0	4	0		l= normal
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		m= perl
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		n= phf
0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0		o= pod
0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	14	0	23	0	0	0	0	0		p= portsweep
0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	3	0	0	0	0	0	0		q= rootkit
0	0	0	0	0	1	0	0	0	3	0	11	0	0	0	16	0	763	0	0	0	0	0		r= satan
0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	3689	0	0	0	0		s= smurf
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		t= spy
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	0	0		u= teardrop
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0		v= warezclient
0	0	0	0	0	1	0	0	2	0	0	2	0	0	0	0	0	0	0	0	0	0	10		w= warezmaster

Tabla 5.34 Matriz de confusión con 10 atributos a 10 folds.

5.10.4 Utilizando 10 atributos y split sample del 66%

Este modelo genero el 99.5% de efectividad en la clasificación que se muestra en la tabla 5.35, siendo el que mejores resultados a mostrado, muy pocos los errores cometidos en este modelo, además de que el tiempo que tomo generarse fue muy bueno pues lo hizo en solo 4.05 seg.

	No. De Instancias	% de efectividad
Instancias correctas	7947	99.5 %
Instancias incorrectas	40	0.5 %
Total	7987	100%

Tabla 5.35 Resultados obtenidos con 10 atributos y split simple 66%.

Este ha sido el modelo que mejor se ajusta a los resultados que buscamos pues cumple en cuanto a la buena clasificación y en cuanto al tiempo de generar dicho modelo. Cabe tomar en cuenta que en este caso solo clasifico un aparte del conjunto total de instancias, aun así este es el modelo satisface nuestras expectativas.

5.10.5 Comparación de resultados Random Forest

La comparación de estos resultados se muestran en la tabla 5.36 y han sido los mejores tanto en el tiempo de clasificación como en efectividad, los mejores resultados obtenido fueron del 99.5% muy superior a lo que se buscaba, sin lugar a dudas los resultados son mas satisfacientes con la selección de atributos.

	Instancias clasificadas correctamente (%)	**Tiempo para construir modelo (s)**
Utilizando 42 atributos		
Validación Cruzada a 10 Folds	98.71	20.6
Split sample del 66%	98.54	9.57
Utilizando 10 atributos		
Validación Cruzada a 10 Folds	99.3	5.06
Split sample del 66%	99.5	4.05

Tabla 5.36 Comparación de resultados obtenidos Random Forest.

Aunque los resultados son muy parecidos del modelo anterior hay una ligera diferencia entre el tiempo y el porcentaje de instancias clasificadas correctamente usando los 42 atributos, siendo este superior, definitivamente una intrusión puede ser clasificada de manera eficiente con ayuda de este modelo y mediante la técnica de Split sample, pues en poco tiempo logramos resultados convincentes.

Capítulo 6

Resultados

Este capitulo mostrara de manera mas detallada los resultados obtenidos en cada experimento a fin de notar las diferencias y similitudes entre ellos, esto se hará dependiendo de la técnica de validación utilizada.

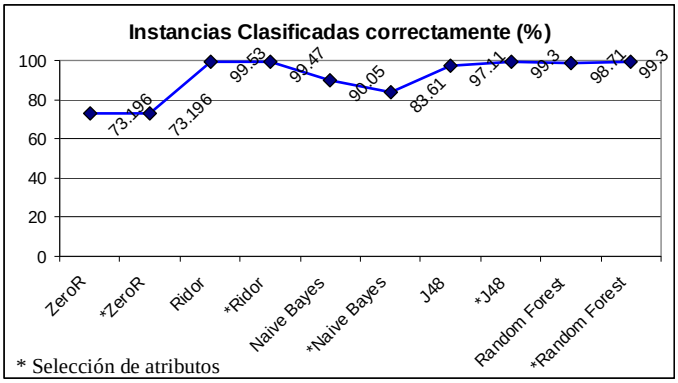
Los resultados obtenidos con la validación cruzada a 10 folds serán los primeros a comparar para ellos la tabla 6.1 mostrara el porcentaje de instancias clasificadas correctamente y el tiempo en que se construyo el modelo.

Algoritmo	Instancias clasificadas correctamente (%)	Tiempo en construir modelo (s)
ZeroR	73.196	1
*ZeroR	73.196	1
Ridor	99.53	16.09 min
*Ridor	99.47	159
Naive Bayes	90.05	109
*Naive Bayes	83.61	38
J48	97.11	22.83
*J48	99.3	5.3
Random Forest	98.71	20.6
*Random Forest	99.3	2.06

* Selección de atributos **Tabla 6.1 Resultados obtenidos con validación a 10 folds.**

Como se puede apreciar en la tabla 6.1 los mejores resultados son obtenidos con los modelos basados en reglas con una efectividad del 99.53% pese a ellos no se considera el mejor pues el tiempo de espera es muy superior a los demás tardando minutos en generar la clasificación mientras que todos los demás se clasifican en segundos. Los peores resultados se obtuvieron con el modelo ZeroR que como se menciono solo se basa en el atributo con mayor numero de apariciones.

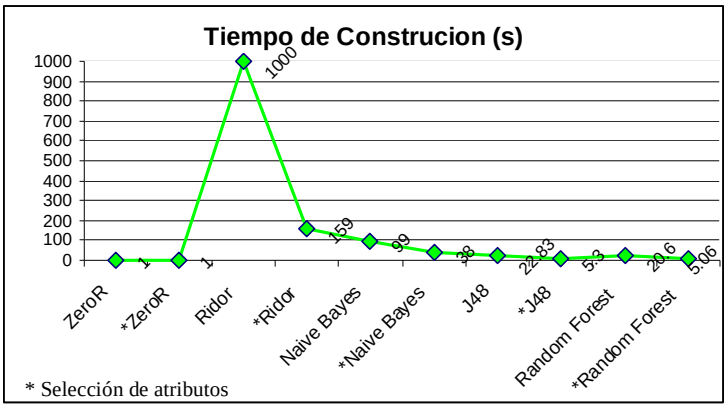
La grafica 6.1 muestra que pese a la selección de atributos algunos modelos no mejoraron en mucho o en nada su clasificación



Grafica 6.1 Porcentaje de instancias clasificadas correctamente a 10 folds.

Los tiempos en los que se construyeron los modelos también muestran la eficiencia de estos, pues se logra apreciar que el modelo construido con el algoritmo Ridor fue por mucho el que mas tiempo se llevo en clasificar, con mas de 16 min, mientras que el de menor tiempo fue el modelo ZeroR con 1 seg. La comparación se muestra en la grafica 6.2.

Vemos pues que los modelos generados por árboles dieron muy buenos resultados. El modelo que mejor resultados brindo fue Random Forest utilizando selección de atributos, su tiempo fue muy bueno pues esta arriba del que mejor tiempo hizo en clasificar. Los demás modelos están por encima de este pero sin tener el porcentaje de aciertos que este.



Grafica 6.2 Comparación de tiempo de los modelos con validación a 10 folds.

Para la validación con Split Sample 66% mostrada en la tabla 6.2 los resultados obtenidos son claros, la que mejores resultados tuvo fue con la técnica Ridor sin embardo el tiempo de construcción fue muy superior son 11:52 min. , demasiado tiempo tomando en cuenta que las instancias podrían ser mayores a las que se clasificaron, sin embargo la técnica de Random Forest con menor tiempo obtuvo casi el mismo resultado, siendo este el mas indicado para esta clasificación.

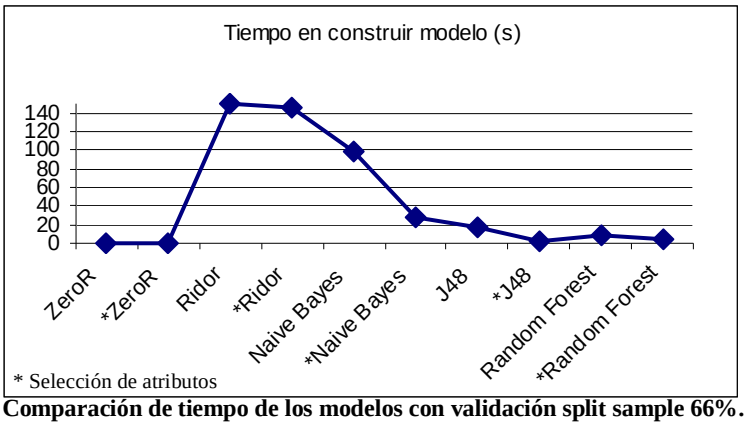
Se puede apreciar que el peor clasificador fue ZeroR con solo 72.48% aunque el tiempo de construcción fue el mejor.

Algoritmo	Instancias clasificadas correctamente (%)	Tiempo en construir modelo (s)
ZeroR	72.48	1
*ZeroR	72.48	1
Ridor	99.56	11.52 min.
*Ridor	99.3	145
Naive Bayes	89.15	98
*Naive Bayes	86.07	28
J48	97.05	16.74
*J48	99.27	2.1
Random Forest	98.54	9.57
*Random Forest	99.5	4.05

Tabla 6.2 Resultados obtenidos con validación split sample 66%.

* Selección de atributos

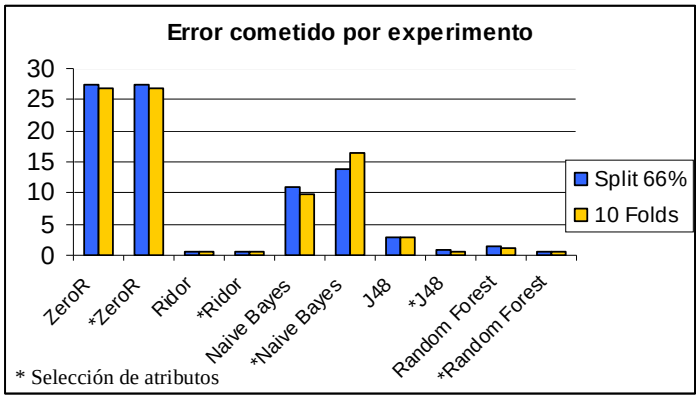
El tiempo de construcción también nos da un claro criterio para evaluar los resultados pues se muestra una clara diferencia entre Ridor que fue el mejor clasificador con el Random Forest que fue nuestro modelo seleccionado, la diferencia aproximada es de 11 minutos. Esto en parte se debió a la selección de atributos, pues solo utilizamos 10 de los 42, reduciendo el tiempo en gran medida. Estos resultados se muestran detalladamente en la grafica 6.3.



Grafica 6.3 Comparación de tiempo de los modelos con validación split sample 66%.

Otra manera en como podemos comparar los resultados es mediante los errores cometidos en cada experimento, para esto utilizaremos la grafica 6.4 donde se muestra los errores cometidos de cada algoritmo con las diferentes técnicas empleadas.

Es notorio el porcentaje de error cometido de la clasificación con Ridor y Random Forest, mientras que el error de ZeroR ha sido el mayor que se registro en estos experimentos.



Grafica 6.4 Error cometido en cada modelo.

Capitulo 7

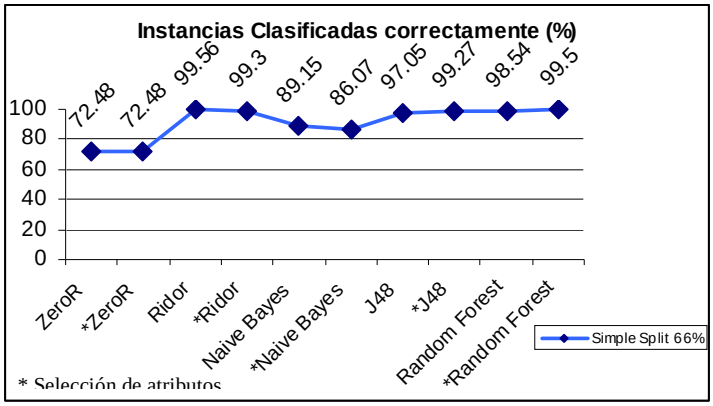
Conclusiones

Después de haber realizado un estudio exhaustivo y haber llegado a ciertas conclusiones, hay que destacar que WEKA da la posibilidad de elegir los atributos más importantes a la hora de construir los modelos de clasificación.

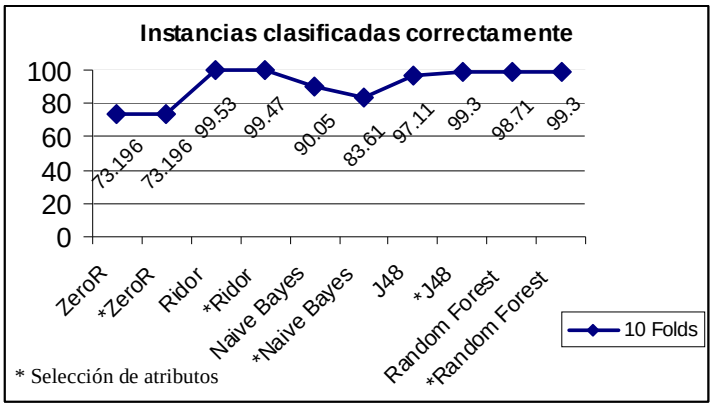
Por lo que respecta a la minería de datos y al descubrimiento del conocimiento (KDD) se observó que contribuyeron a la toma de decisiones tácticas y estratégicas; proporcionando un sentido automatizado para la generación del conocimiento y por consiguiente ayudar a tomar decisiones acertadas en el campo de las intrusiones de intrusos.

Para dicha clasificación weka ha sido muy eficiente, pues los resultados que obtuvimos mediante sus clasificadores sobrepasaron el 90% que se buscaba para determinar con solides que se pueden detectar intrusiones en base a los patrones obtenidos en la clasificación.

En las grafica 7.1 y 7.2 se muestran los resultados de los experimentos y su porcentaje de efectividad de acuerdo al tipo de clasificación y al método empleado.

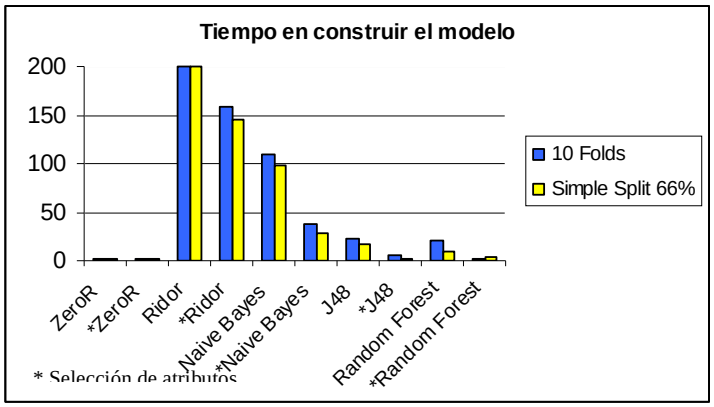


Grafica 7.1 Porcentaje de instancias clasificadas con Simple Split 66%



Grafica 7.2 Porcentaje de instancias clasificadas con 10 folds.

Como se puede apreciar en las graficas los resultados fueron muy similares pues la mayoría de ellos sobrepasan el porcentaje buscado, sin embargo algo en lo que se nota mayor diferencia es en los tiempos de construcción de modelo.



Grafica 7.3 Comparación del tiempo en construir los modelos.

El tiempo fue un factor a considerar en los resultados finales pues en el caso de la clasificación con Ridor el tiempo fue excesivo en comparación con los demás modelos, pues utilizo cerca de 12 minutos para terminar con la clasificación con selección de atributos, lo que hace poco eficiente al modelo pues solo se clasifico el 10 % del total de las instancias estos resultados se aprecian en la grafica 7.3.

Como se pudo notar en las graficas los mejores resultados se obtuvieron con la selección de atributos, por eso se realizo un pequeño análisis mostrado den la tabla 7.1 sobre porqué WEKA piensa que algunos de esos atributos son los mejores:

protocol_type	El tipo de protocolo es bastante importante, puesto que unos protocolos son más vulnerables que otros.
service	Con el servicio ocurre lo mismo que con el tipo de protocolo.
src_bytes	El número de bytes de datos de fuente a destino es significativo puesto que normalmente los ataques suceden con paquetes pequeños.
dst_bytes	Por la mismo que el anterior el número de bytes de datos de destino a la fuente también es importante.
flag	Estado de la conexión (SF, S1, REJ…), muchos ataques se suelen realizar cuando se está en un estado conocido.
wrong_fragment	Cuando el número de fragmentos erróneos es elevado, puede deberse a que se está produciendo un ataque.
count	Es posible que si alguien esté intentando conectarse a una maquina puede deberse a que está intentando entrar o atacar esa máquina.

Tabla 7.1 Atributos de mayor peso en la clasificación.

Algo importante de señalar en base a los resultados obtenidos es que la naturaleza de cada uno de los algoritmos de aprendizaje influyó en la precisión de la clasificación de un mismo conjunto de datos.

En cuanto al algoritmo de clasificación, destaca por su sencillez y su alto porcentaje de aciertos el algoritmo basado en árboles Random Forest.

Por último, si se tuviese que seleccionar un segundo clasificador, sería el J48 intentando eliminar algunos atributos para poder así obtener resultados más rápidamente o poder complicar un poco más otros algoritmos de clasificación.

Bibliografía

- Morgan Kaufmann, *Data Mining Practical Machine Learning Tools and Techniques*, 2nd edición, USA, Morgan Kaufmann Publishers 2005.

- Mehmed Kantardzic, *Data Mining - Concepts, Models, Methods, and Algorithms*, 1er. Edicion, John Wiley & Sons 2003.

- Fayyad, Usama M. Grinstein, Georges G. Wierse, Andreas., *Information Visualization In Data Mining And Knowledge Discovery*, San Francisco: MK/Morgan Kaufmann Publishers, c2002.

- Dr. Ivo Humberto Pineda Torres, *Minería de datos y almacenes de datos*, Puebla 2007.

- University of California, Irvine. KDD Cup 1999 Data http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

- Lincoln Laboratory. Short Attack Descriptions - 1999 DARPA Intrustion Detection Evaluation http://www.ll.mit.edu/IST/ideval/docs/1999/attackDB- ShortForm.html

Publicaciones:

- María N. Moreno García, Luis A. Miguel Quintales, Francisco J. García Peñalvo y M. José Polo Martín, *Aplicación de técnicas de minería de datos en la construcción y validación de modelos predictivos y asociativos a partir de especificaciones de requisitos de software*, Universidad de Salamanca. Departamento de Informática y Automática 2004

- ACM Inc. ACM Computing Classification System [V.1998] http://www.acm.org/class/1998/

- Documentación a cerca de la herramienta WEKA. http://salaam.cs.buap.mx/DM/AUDIOVISUAL-PRESENTACIONES/

- Página de documentación Weka. Universidad de Waikato. http://www.cs.waikato.ac.nz/~ml/weka/index.html

- SCALAB, Grupo de Invetigación. Software de Aprendizaje Automático - Tutorial Weka. http://scalab.uc3m.es/~docweb/aa/software.html

- Wikipedia. Ataque de denegación de servicio http://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio