

**Descubrimiento de conocimiento mediante la participación de dos  
competencias en Kaggle**

**Bryan Valencia Suárez**

**Presentado como informe de trabajo final en la asignatura de  
Descubrimiento de Conocimiento en Bases de Datos**

**Universidad Politécnica de Madrid  
Escuela Técnica Superior de Ingenieros Informáticos  
Máster en Software y Sistemas  
Madrid, España  
15 de enero de 2018**

# Contenido

Introducción.....	3
1. Titanic: Machine Learning from Disaster.....	4
1.1. Objetivos de la competición y descripción del dominio .....	4
1.2. Descripción de los datos.....	4
1.3. Limpieza y pre procesamiento de los datos .....	5
1.4. Reducción y proyección de datos .....	6
1.4.1. Eliminación de atributos.....	6
1.4.2. Generación de nuevos atributos .....	7
1.5. Selección del algoritmo de minería de datos .....	11
1.6. Aplicación del algoritmo de minería de datos .....	12
1.7. Interpretación y envío a Kaggle.....	14
2. Digit Recognizer.....	17
2.1. Objetivos de la competición y descripción del dominio .....	17
2.2. Descripción de los datos.....	17
2.3. Reducción y proyección de datos.....	18
2.4. Selección del algoritmo de minería de datos .....	20
2.5. Aplicación del algoritmo de minería de datos .....	20
2.6. Interpretación y envío a Kaggle.....	22
2.6.1. Aplicación del algoritmo RandomTree en Weka.....	22
2.6.2. Aplicación del algoritmo RandomForest en Weka.....	23
2.6.3. Aplicación del algoritmo RandomForest en Python.....	24
Conclusiones .....	25
Referencias.....	27

# Descubrimiento de conocimiento mediante la participación de dos competencias en Kaggle

Bryan Valencia Suárez

## Introducción

Mediante el proceso de Descubrimiento de Conocimientos de Bases de Datos (KDD) se puede revelar y extraer información útil (conocimiento) de conjuntos de datos para ser aplicada en el campo correspondiente y mejorar la calidad de la toma de decisiones [1]. Algunas veces a este proceso se le denomina Minería de Datos (Data Mining), pero realmente Data Mining es sólo uno de los pasos dentro de todas las tareas que son necesarias para llevar a cabo esta actividad.

En el presente documento, se realiza el proceso KDD a dos conjuntos de datos de dos competencias presentadas por una plataforma para competencias en Data Science llamada Kaggle [2]. En este sitio web, mineros de datos y estadísticos compiten para producir los mejores modelos para predecir y describir los conjuntos de datos cargados por empresas y usuarios.

Las dos competencias elegidas están en la categoría de “Getting Started”, lo que significa que son retos para aquellas personas que están iniciando su experiencia en Data Science. El nombre de los dos concursos es “Titanic: Machine Learning from Disaster” y “Digit Recognizer”. Posteriormente en este documento, se exponen todas las tareas llevadas a cabo para extraer conocimiento de estos conjuntos de datos y alcanzar los objetivos de cada competencia.

También se ha utilizado Weka [3], que es un paquete de software de aprendizaje automático escrito en Java, desarrollado en la Universidad de Waikato, Nueva Zelanda. Weka apoya el proceso de KDD ofreciendo algunos módulos para la visualización gráfica de los datos, la selección y limpieza de los mismos, la elección y aplicación de algoritmos de Data Mining, entre otros. Es software libre licenciado bajo la Licencia Pública General de GNU.

Por otra parte, la metodología tradicional de KDD propone que este proceso debe desarrollarse mediante los siguientes pasos [4]:

1. Aprendizaje del dominio de la aplicación
2. Creación de un conjunto de datos de destino
3. Limpieza y preprocesamiento de datos
4. Reducción y proyección de datos
5. Selección de la función de minería de datos
6. Selección del algoritmo de minería de datos
7. Data Mining
8. Interpretación
9. Uso del conocimiento descubierto

En los proyectos que aquí se presentan, este método ha sido ajustado a la competencia y a las necesidades del dominio de cada problema, por lo que algunas fases se omiten y otras se combinan.

## **1. Titanic: Machine Learning from Disaster**

### **1.1. Objetivos de la competición y descripción del dominio**

El RMS Titanic fue un barco británico que se hundió el 15 de abril de 1912 durante su viaje inaugural, después de colisionar con un iceberg, matando alrededor de 1500 de 2224 pasajeros y tripulantes. Esta tragedia que es considerada como uno de los mayores naufragios de la historia, conmocionó a la comunidad internacional y condujo a mejores regulaciones de seguridad para los buques.

Una de las razones por las que el naufragio provocó tal pérdida de vidas fue que no había suficientes botes salvavidas para los pasajeros y la tripulación. Aunque hubo algún elemento de suerte involucrado en sobrevivir al hundimiento, algunos grupos de personas tenían más probabilidades de sobrevivir que otros, como las mujeres, los niños y la clase alta.

En esta competencia expuesta por Kaggle, los participantes deben analizar qué tipo de personas probablemente sobrevivieron. En particular, se deben aplicar técnicas para predecir, a partir de un conjunto de datos de los pasajeros, cuáles de ellos sobrevivieron a esta tragedia y cuáles no.

### **1.2. Descripción de los datos**

Kaggle provee dos conjuntos de datos para realizar el procedimiento de Minería de datos. Los conjuntos de datos son listas de pasajeros del Titanic con algunos atributos descriptores.

Uno de los conjuntos de datos, denominado “train”, contiene un listado de 891 pasajeros, el segundo, llamado “test”, tiene 418 registros.

El conjunto “train”, como su nombre lo indica, debe ser usado para construir el modelo mediante el algoritmo de minería de datos que se seleccione. Estos datos contienen en uno de los atributos la respuesta por cada pasajero, es decir, indica si el pasajero sobrevive o no al naufragio. El modelo estará basado en las características dadas de cada pasajero, como por ejemplo, edad, género, clase, entre otros.

Por su parte, “test”, no contiene la respuesta (es decir, no indica si cada pasajero sobrevive o no al naufragio) y es usado para verificar el modelo construido con los datos de entrenamiento.

Después de aplicar el modelo al conjunto de datos “test”, éste se publica en el sistema de Kaggle para validar cuál fue el porcentaje de aciertos que el modelo construido generó. Sólo mediante el proceso de publicación de Kaggle se puede obtener los valores de acierto para el conjunto de datos “test”.

Los atributos de los conjuntos de datos son los siguientes:

Nombre atributo	Descripción	Opciones (cuando es nominal)
survival	Indica si el pasajero sobrevive o no.	0=No 1=Sí
pclass	Indica la clase del ticket que fue adquirida por el pasajero. Existen tres clases, donde la primera representa el ticket con mejores condiciones, y la tercero con peores.	1=Primera clase (alta) 2=Segunda clase (media) 3=Tercera clase (baja)
sex	El sexo del pasajero	female male
age	Edad del pasajero. Si la edad del pasajero es mejor a un año, esta se indica en un número fraccionario. Si la edad es estimada, esta se indica en la forma xx.5.	
sibsp	Número de hermano y cónyuges a bordo del Titanic. El conjunto de datos define las siguientes relaciones familiares: Hermano = hermano, hermana, hermanastro, hermanastra Cónyuge = esposo, esposa (las amantes y los novios fueron ignorados)	
parch	Número de padres e hijos a bordo del Titanic. El conjunto de datos define las siguientes relaciones familiares: Padre = madre, padre Hijo = hija, hijo, hijastra, hijastro Algunos niños viajaron solo con una niñera, por lo tanto parch = 0 para ellos.	
ticket	Número del ticket	
fare	Tarifa pagada por el pasajero	
cabin	Número de cabina	
embarked	Puerto de embarcación	C=Cherbourg Q=Queenstown S=Southampton

### 1.3. Limpieza y pre procesamiento de los datos

El proceso realizado para los conjuntos de datos de la competición del Titanic comenzó con un análisis para identificar ruido y datos faltantes.

El atributo edad contenía originalmente muchos campos vacíos, por lo que se toma la decisión de completarlos con la edad promedio de todos los pasajeros de los que se conocía esta información.

Para generar una edad promedio más precisa, se hizo uso de los títulos que estaban dados en los nombres de los pasajeros, es decir Mr., Miss y Mrs., con el fin de tener este valor en cuenta ya que las mujeres con tratamiento Miss., tenían una edad promedio menor que las mujeres con tratamiento Mrs., de esta manera se obtuvieron las siguientes edades promedio:

Tratamiento	Descripción	Edad promedio
Miss.	Mujeres solteras	21.8
Mrs.	Mujeres casadas	35.7
Mr.	Hombres solteros y casados	32

Para el caso de los hombres, al no existir una diferenciación con el tratamiento que permitiera discriminar la edad, se hizo un promedio de todas las edades.

Posteriormente, teniendo en cuenta el tratamiento de los pasajeros sin edad especificada, se completó la misma con los valores promedios ya mencionados.

Por otra parte, para que el archivo fuera leído correctamente por Weka, los campos vacíos de otros atributos del conjunto de datos fueron completados con el símbolo “?”.

Estos procedimientos se llevaron a cabo tanto en el archivo de entrenamiento como en el de test.

Así mismo, al archivo “test” se le adicionó la columna “survived” y para cada registro se agregó el símbolo “?”. Sin esta columna, Weka no reconoce que el archivo de entrenamiento y el de test tienen la misma estructura y no hubiera permitido realizar las clasificaciones.

En esta etapa no se eliminaron registros de los conjuntos de datos.

## **1.4. Reducción y proyección de datos**

### **1.4.1. Eliminación de atributos**

La eliminación de características es un proceso que se denomina técnicamente Feature Selection. Es especialmente útil en conjuntos de datos de alta dimensionalidad, es decir aquellos con una gran cantidad de atributos. En este caso, el listado de pasajeros sólo contiene 10 características y alrededor de 800 registros, pero de igual forma se decide eliminar algunas características pensando en la optimización del tiempo de ejecución del algoritmo de Data Mining.

Dentro del proceso de reducción de la dimensionalidad del conjunto de datos, se identificaron aquellas características que no aportaban valor al proceso de Data

Mining, es decir, aquellas que no facilitaban la clasificación de cada pasajero entre sobreviviente o no.

Por lo tanto, los siguientes fueron los atributos que se eliminaron del conjunto de datos:

Atributo	Razón de eliminación
PassengerId	En el conjunto de datos existe un valor número que es la identificación única otorgada a cada pasajero. Realmente este es un valor autoincremental creado para manejar el conjunto de datos pero que no tiene relación con los pasajeros, por lo que se decide eliminar previamente a la aplicación del algoritmo de Data Mining.
Name	El nombre del pasajero fue eliminado del conjunto de atributos porque no aportaba valor a la clasificación. El hecho de que una persona tenga un nombre u otro no fue considerado como un indicador de su supervivencia al naufragio del Titanic. El campo nombre contenía, además, el tratamiento (Mr., Miss., Mrs.) que fue extraído en otro atributo que se explicará posteriormente en este documento
Ticket	Con el número del ticket del pasajero, no había una homogeneidad en los datos registrados en esta característica ya que algunos tickets contenían solamente valores numéricos, y otros eran una combinación de letras y números, pero no fue posible hallar una relación útil en este campo que aportara valor al proceso. Además de eso se pensó en algún momento que el ticket podría dar información de la calidad de pasaje comprado, pero esta era un dato que ya se tenía con el atributo "class".

#### 1.4.2. Generación de nuevos atributos

Como se mencionó previamente, el campo "name" del conjunto de datos de los pasajeros también contenía el tratamiento de la persona. Ejemplo:

Heikkinen, Miss. Lain

La segunda palabra es el tratamiento, que en este caso es "Miss.". Aunque el nombre de los pasajeros fue eliminado como atributo, se creó una nueva característica llamada "title" para almacenar esta información. Este es un atributo nominal con las siguientes opciones:

- Miss
- Mr
- Mrs
- Master
- Other

La opción “other” es utilizada para aquellos pasajeros a los que no se les indicó un tratamiento o éste no se encontraba en la opciones creadas.

El tratamiento “Master” en aquella época se utilizaba para referirse al hermano varón mayor entre un grupo de hijos, lo que estaba otorgándole al sistema de Data Mining un descriptor importante sobre la edad y sexo de los pasajeros.

Por otra parte, el atributo “cabin” estaba compuesto por dos datos: la cubierta y la cabina que fue asignada a cada pasajero. Ejemplo:

E46

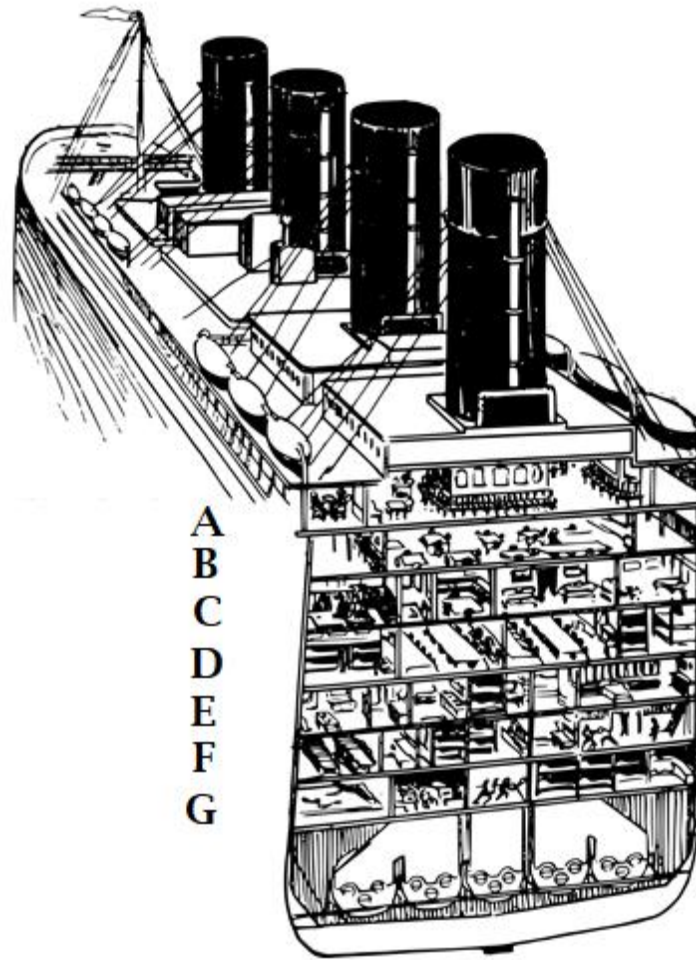
En este dato, el pasajero tenía asignada la cabina No. 46 en la cubierta (Deck en inglés) “E”.

Nótese en la siguiente lista de pasajeros sobrevivientes que proporcionalmente, sobrevivieron más pasajeros de primeras clases [5]:

<b>Clase</b>	<b>Cantidad de pasajeros</b>	<b>Pasajeros sobrevivientes</b>	<b>Porcentaje</b>
Primera clase	324	201	62%
Segunda clase	277	118	43%
Tercera clase	708	181	26%
Integrantes de la tripulación	885	212	24%
Carteros y músicos	13	0	0%
<b>Gran total</b>	<b>2207</b>	<b>712</b>	<b>32%</b>

Las cubiertas estaban enumeradas de la ‘A’ a la ‘G’, siendo la cubierta ‘A’ la más cercana a la cubierta más superior (que contenía los botes salvavidas) y la cubierta G una de las más inferiores (ver Figura 1).





***Figura 1. Diagrama de la sección transversal del Titanic [6]***

Aunque había servicios de primera y segunda clase en casi todas las cubiertas, es decir salas de fumadores, bibliotecas, restaurantes, entre otros, las cubiertas A, B y C concentraban el mayor número de habitaciones y lugares destinados para los pasajeros de mejor clase.

Por lo anterior, se decide crear dos nuevos atributos nominales para almacenar de manera independiente la cubierta y el número de la cabina. Estas características fueron nombradas “decks” y “cabinNum”, respectivamente. En la siguiente tabla pueden verse las opciones de cada nuevo atributo.

Se identificó que un ticket podía pertenecer a varias personas, y por tanto el atributo “fare” (tarifa) indicaba el monto total pagado, pero no el valor por persona, por lo que los datos no eran homogéneos, ya que para aquellas personas que viajaban solas, se indicaba el valor por persona, pero para los que no, se estaba indicando el monto total del ticket.

Por ejemplo, el ticket “W./C. 6608”, pertenecía a las siguientes 5 personas pertenecientes a la familia Ford:

'Ford, Mr. William Neal'  
 'Ford, Miss. Robina Maggie Ruby'  
 'Ford, Miss. Doolina Margaret Daisy'  
 'Ford, Mrs. Edward (Margaret Ann Watson)'

Para solucionarlo, mediante una aplicación realizada en Java, se dividió el monto del billete en el total de personas que lo compartían. En el anterior ejemplo, el monto total pagado fue de 34.375, pero en realidad el valor por persona fue 6.875, que fue el resultado de dividir el valor entre los 5 integrantes de la familia Ford.

El último atributo generado fue un proceso de Feature Extraction en el que dos características se convirtieron en una sola. El conjunto de datos original contenía un atributo para indicar el número de hermanos y cónyuges (sibsp) y otro (parch) para el número de padres e hijos a bordo del Titanic por cada pasajero. La estrategia desarrollada aquí fue sumar estos dos valores y convertirlos en uno sólo llamado tamaño de familia (fsize) ya que se consideró que tanto padres, como hermanos y cónyuges estaban en un mismo núcleo familiar y que la intención de permanecer juntos y/o de salvar a ciertos miembros particulares de la familia podría influir en la supervivencia de los pasajeros.

Por lo tanto, los atributos que finalmente se conservaron en los archivos de training y test fueron:

Nombre atributo	Tipo	Opciones
pclass	Nominal	1, 2, 3
sex	Nominal	Female, Male
age	Numérico	
fsize	Numérico	
farePP	Numérico	
embarked	Nominal	C, Q, S
decks	Nominal	'A','B','C','D','E','F','G','T'
cabinNum	Nominal	'0','1','10','101','102','103','104','106','11','110','111','118','12','121','123','124','125','126','128','14','148','15','16','17','18','19','2','20','21','22','23','24','25','26','28','3','30','31','32','33','34','35','36','37','38','39','4','40','41','42','44','45','46','47','48','49','5','50','52','54','56','58','6','63','65','67','68','69','7','70','71','73','77','78','79','8','80','82','83','85','86','87','9','90','91','92','93','94','95','99'
title	Nominal	"Mr","Mrs","Miss","Master","Other"
survived	Nominal	1, 0

Como último proceso antes de la selección del algoritmo de Data Mining, el archivo CSV que fue proporcionado por Kaggle, debió ser convertido a un archivo tipo ARFF que es el formato aceptado por Weka.

Para realizar esta conversión, se cambió manualmente la extensión del archivo y se agregaron los siguientes encabezados:

```
@attribute pclass {1,2,3}
@attribute sex {female,male}
@attribute age numeric
@attribute fsize numeric
@attribute farePP numeric
@attribute embarked {C,Q,S}
@attribute decks { 'A','B','C','D','E','F','G','T' }
@attribute cabinNum{
'0','1','10','101','102','103','104','106','11','110','111','118','12','121','123','124','125',
'126','128','14','148','15','16','17','18','19','2','20','21','22','23','24','25','26','28','3',
30,'31','32','33','34','35','36','37','38','39','4','40','41','42','44','45','46','47','48','49',
5,'50','52','54','56','58','6','63','65','67','68','69','7','70','71','73','77','78','79','8','80',
82,'83','85','86','87','9','90','91','92','93','94','95','99' }
@attribute title { "Mr","Mrs","Miss","Master","Other" }
@attribute survived {0,1}
```

Para que Weka pueda generar los valores de respuesta en el archivo de “test” éste debe contener exactamente la misma estructura que el archivo “train”, por lo que debe crearse un atributo “survived” e indicar el símbolo “?” en cada uno de los registros.

### 1.5. Selección del algoritmo de minería de datos

Después de la realización de algunas pruebas con algunos algoritmos en Weka, se evidenció que el modelo que generaba mejores resultados fue el llamado J48.

J48 es en realidad una implementación Open Source en lenguaje de programación Java del algoritmo C4.5 en la herramienta Weka.

El algoritmo C4.5 fue desarrollado por JR Quinlan en 1993, como una mejora del algoritmo ID3 que desarrolló en 1986. El algoritmo C4.5 genera un árbol de decisión a partir de los datos mediante particiones realizadas recursivamente. El árbol se construye mediante la estrategia de profundidad-primer (depth-first). El algoritmo

considera todas las pruebas posibles que pueden dividir el conjunto de datos y selecciona la prueba que resulta en la mayor ganancia de información.

Para cada atributo discreto, se considera una prueba con  $n$  resultados, siendo  $n$  el número de valores posibles que puede tomar el atributo. Para cada atributo continuo, se realiza una prueba binaria sobre cada uno de los valores que toma el atributo en los datos. En cada nodo, el sistema debe decidir cuál prueba escoge para dividir los datos.

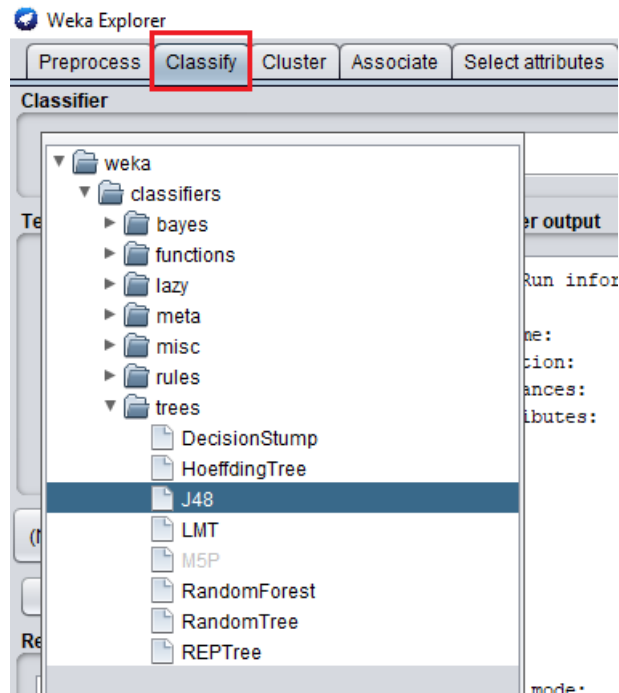
C4.5 crea árboles de decisión desde un grupo de datos de entrenamiento usando el concepto de entropía de información. Los datos de entrenamiento son un grupo de ejemplos ya clasificados.

En cada nodo del árbol, C4.5 elige un atributo de los datos que más eficazmente dividen el conjunto de muestras en subconjuntos enriquecidos en una clase u otra. Su criterio es el normalizado para ganancia de información (diferencia de entropía) que resulta en la elección de un atributo para dividir los datos. El atributo con la mayor ganancia de información normalizada se elige como parámetro de decisión. El algoritmo C4.5 divide recursivamente en sublistas más pequeñas [7].

Con este algoritmo es posible trabajar con valores continuos para los atributos, separando los posibles resultados en 2 ramas  $A_i \leq N$  y  $A_i > N$ . Los árboles son menos frondosos, ya que cada hoja cubre una distribución de clases no una clase en particular.

## **1.6. Aplicación del algoritmo de minería de datos**

La aplicación del algoritmo en Weka consiste en la selección del archivo ARFF que se preparó en fases anteriores y, en la pestaña "Classify", que se resalta en la siguiente imagen, se selecciona el algoritmo a aplicar.



Antes de aplicar el algoritmo al conjunto de test, se entrenó el modelo en el conjunto “train”, obteniendo el siguiente árbol de decisión

```
sex = female
| farePP <= 9.84
| | fsize <= 4
| | | decks = A: 1 (0.0)
| | | decks = B: 1 (0.0)
| | | decks = C: 1 (0.0)
| | | decks = D: 1 (14.0)
| | | decks = E: 0 (2.0/1.0)
| | | decks = F
| | | | embarked = C: 1 (23.0/8.0)
| | | | embarked = Q
| | | | | farePP <= 7.67: 0 (2.0)
| | | | | farePP > 7.67
| | | | | | fsize <= 2: 1 (27.0/4.0)
| | | | | | fsize > 2: 0 (3.0/1.0)
| | | | | embarked = S
| | | | | | farePP <= 7.67: 1 (18.0/5.0)
| | | | | | farePP > 7.67: 0 (39.0/15.0)
| | | | decks = G
| | | | | farePP <= 5.3: 0 (2.0)
| | | | | farePP > 5.3: 1 (2.0)
| | | | decks = T: 1 (0.0)
| | | fsize > 4: 0 (28.0/4.0)
| | farePP > 9.84: 1 (154.0/9.0)
sex = male
| title = Mr: 0 (517.0/81.0)
```

	title = Mrs: 0 (0.0)
	title = Miss: 0 (0.0)
	title = Master: 1 (40.0/17.0)
	title = Other: 0 (20.0/5.0)

En este árbol de decisión puede notarse que el algoritmo seleccionó el sexo como primer atributo para empezar a clasificar el conjunto de datos. Posteriormente, y en el caso de las mujeres, usa la tarifa por persona, la cubierta, el tamaño de la familia y el lugar de embarque. En el caso de los hombres, usa como segundo clasificador únicamente el tratamiento, indicando, por ejemplo, que los hombres con tratamiento “Master” sobrevivían al naufragio.

### 1.7. Interpretación y envío a Kaggle

Como puede verse a continuación, se clasificaron bien 725 de los 891 pasajeros que contiene el archivo de entrenamiento, lo que representa un 81,3% del total.

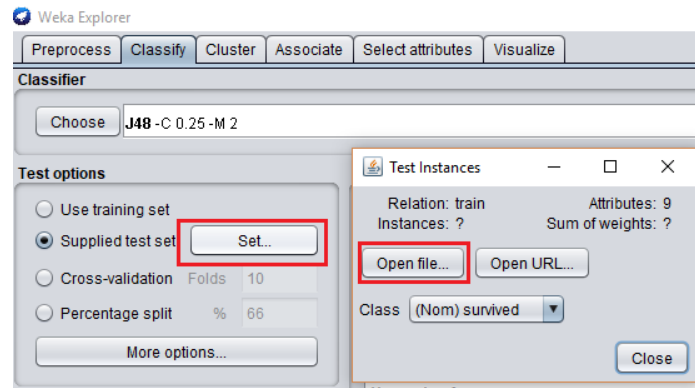
Correctly Classified Instances	725	81.3692 %
Incorrectly Classified Instances	166	18.6308 %
Kappa statistic	0.5922	
Mean absolute error	0.2762	
Root mean squared error	0.3824	
Relative absolute error	58.3798 %	
Root relative squared error	78.6398 %	
Total Number of Instances	891	

La matriz de confusión muestra que exactamente la tercera parte de los pasajeros que sobrevivieron fueron marcados como NO sobrevivientes y para 52 de ellos se indicó sobrevivencia cuando no fue así.

```
a  b  <-- classified as
497 52 | a = 0
114 228 | b = 1
```

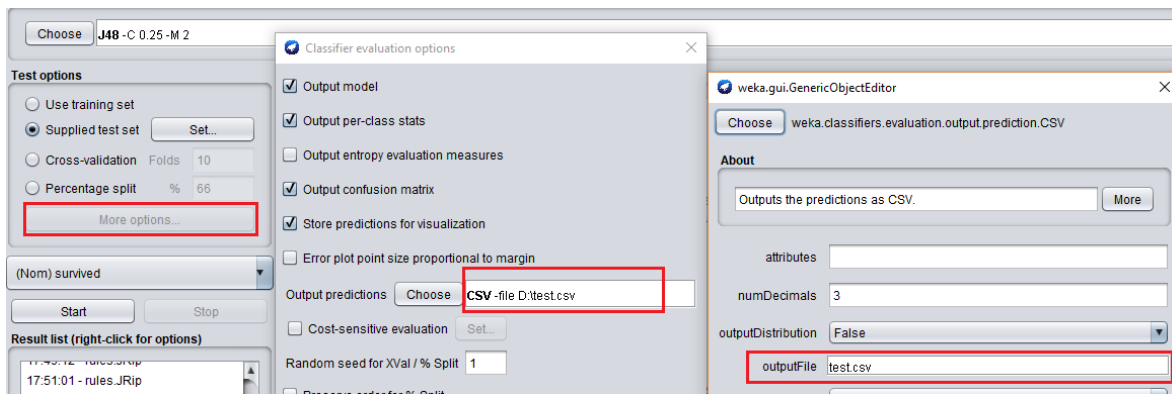
Subsiguientemente, el modelo creado fue aplicado al conjunto de datos “test” que evalúa Kaggle. Generar este archivo en Weka no fue un proceso intuitivo y por lo que en esta sección se detallan los pasos para realizarlo.

En la opción “Supplied test set” se debe dar clic en “set” y posteriormente en la ventana emergente dar clic en “Open File”.



Debe seleccionarse el archivo con el conjunto de datos “test” otorgado por Kaggle. Es muy importante que este archivo, además de tener la extensión .ARFF, debe también contener en uno de sus atributos la característica “survived”, y como para el archivo de test no se contiene el valor de respuesta, debe completarse cada registro con el valor “?” para que sea entendible por Kaggle.

Weka permite configurar un archivo destino para generar las respuestas por cada registro del conjunto de datos de “test” otorgado, para esto debe darse clic en “More options”, clic derecho en el campo de texto de “Output predictions”, “Show properties” y en el campo resaltado más a la izquierda en la siguiente imagen, la ubicación del archivo destino.



Con esto puede darse clic en “start” y el archivo test lucirá así:

```
inst#,actual,predicted,error,prediction
```

```
1,1?,1:0,,0.843
2,1?,2:1,,0.722
3,1?,1:0,,0.843
4,1?,1:0,,0.843
5,1?,2:1,,0.722
6,1?,1:0,,0.843
```

```
•
•
•
```

Esta estructura de archivo no es útil para publicarla en Kaggle, pero ofrece alguna información importante como el porcentaje de predicción para cada registro (atributo 4 “prediction”). El campo que interesa en este caso es el número 3 llamado “predicted”.

Este campo contiene un conjunto de datos de la manera <número de opción>:<opción>. En este caso solo hay dos opciones:

Número de opción	Valor de la opción	Descripción
1	0	El pasajero no sobrevive
2	1	El pasajero sí sobrevive


Para extraer los datos correctos, es entonces necesario tomar el valor de este campo que está después de los puntos, y crear un archivo con un valor auto incremental que empieza en 892 y nombres de atributos específicos como se muestra en este ejemplo:

```
PassengerId,Survived
892,0
893,1
894,0
895,0
896,1
897,0
.
.
.
```

Para extraer la información se utilizaron editores de texto como NotePad y Microsoft Excel. Al final se guarda con una extensión CSV para publicarlo en Kaggle.

En Kaggle se obtuvo un porcentaje de instancias correctamente clasificadas de un 80,3%.


1210 ▲ 42 Bryan Valencia

 0.80382 12 now

Your Best Entry ▲

You advanced 163 places on the leaderboard!

Your submission scored 0.80382, which is an improvement of your previous score of 0.79904. Great job!

 Tweet this!



## 2. Digit Recognizer

### 2.1. Objetivos de la competición y descripción del dominio

En esta competencia, el objetivo es identificar correctamente el dígito que corresponde a cada conjunto de píxeles de miles de imágenes de números escritos a mano. Los posibles valores son los números del 0 al 9.

La métrica de evaluación para este concurso es la precisión de categorización, o la proporción de imágenes de prueba que están clasificadas correctamente. Por ejemplo, una precisión de categorización de 0.97 indica que se han clasificado correctamente todas las imágenes menos el 3%.

### 2.2. Descripción de los datos

Los archivos de datos train.csv y test.csv contienen imágenes en escala de grises de dígitos dibujados a mano, del cero al nueve.

Cada imagen tiene 28 píxeles de altura y 28 píxeles de ancho, para un total de 784 píxeles. Cada píxel tiene un único valor asociado, que indica la luminosidad u oscuridad de ese píxel, con números más altos que significan más oscuros. Este valor de píxel es un número entero entre 0 y 255.

El conjunto de datos de entrenamiento, (train.csv), tiene 785 columnas. La primera columna, llamada "label", es el dígito dibujado por el usuario. El resto de las columnas contienen los valores de píxel de la imagen asociada.

Cada columna de píxeles en el conjunto de entrenamiento tiene un nombre como pixel<x>, donde <x> es un número entero entre 0 y 783, inclusive. Para ubicar este píxel en la imagen, suponga que se ha descompuesto x como  $x = i * 28 + j$ , donde i y j son números enteros entre 0 y 27. Luego, pixel<x> se ubica en la fila i y en la columna j de una matriz de 28 x 28, (indexando por cero).

Por ejemplo, pixel87 indica el píxel que está en la cuarta columna desde la izquierda, y la cuarta fila desde la parte superior, como se resalta en la tabla a continuación.

Visualmente, si se omitiera el prefijo "pixel", los píxeles componen la imagen así:

0	1	2	3	4	5	6	7	...	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	...	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	...	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	...	105	106	107	108	109	110	111
.	.	.	.	.	.	.	.	...	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	...	.	.	.	.	.	.	.
700	701	702	703	704	705	706	707	...	721	722	723	724	725	726	727
728	729	730	731	732	733	734	735	...	749	750	751	752	753	754	755
756	757	758	759	760	761	762	763	...	777	778	779	780	781	782	783

El conjunto de datos de prueba, (test.csv), contiene 28000 registros y es el mismo que el conjunto de entrenamiento, excepto que no contiene la columna "etiqueta". Por otra parte, el conjunto de entrenamiento tiene 42000 registros.

### 2.3. Reducción y proyección de datos

Como se presentó previamente, el conjunto de datos tiene un total de 784 características que representan cada píxel en la imagen y decenas de miles de registros, esto hizo que al momento de intentar usar los algoritmos de clasificación de Weka, éste se tardara incluso horas en generar un resultado.

Debido a la anterior, se diseñó una estrategia para realizar una reducción de dimensionalidad pensando en conservar de alguna manera las características de cada imagen. La estrategia planteada fue convertir cada una de las imágenes de 28x28px a imágenes de 14x14px, para lo cual se realizó el promedio de cada 4 píxeles continuos almacenados por imagen.

Cuatro píxeles continuos no significaba 4 números continuos en cada registro del conjunto de datos, ya que era importante tener en mente que cada 28 números correspondían a una línea diferente.

Para entender la reducción realizada, tenga en cuenta que en el siguiente ejemplo, cada cuadrícula de la tabla contiene a su izquierda (con fondo azul) el número o posición del píxel y al lado derecho (con fondo rojo) el valor del mismo.

Nótese cómo los valores de los píxeles 0, 1, 28 y 29, son promediados para obtener un nuevo número (255), como también los píxeles 2, 3, 30 y 31 generan el pixel con el valor 102.

Imagen origen

0	255	1	255	2	100	3	25	...
28	255	29	255	30	30	31	255	...
.								
.								
.								

Imagen resultante

0	255	1	102	...
.				
.				
.				

De esta manera los conjuntos de datos pasaron a tener 196 características.

Para realizar esta reducción, se realizó un programa en Java, que recibía el archivo CSV original propuesto por Kaggle y generaba un archivo CSV con 196 características. El método principal se muestra a continuación:

```
public void generateTestDigitRecognizer() throws FileNotFoundException{
    String csvFile = "resources/digitRecognier/test_original.csv";
    String line = "";
    String cvsSplitBy = ",";
    PrintWriter pw;
    pw = new PrintWriter(new
    File("resources/digitRecognier/test_limpiado.csv"));
```

```

        StringBuilder sb = new StringBuilder();

        for(int j=0; j<196; j++){
            sb.append("pixel"+j);
            if(j<19)
                sb.append(",");
        }
        sb.append('\n');

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
            br.readLine();
            while ((line = br.readLine()) != null) {
                String[] image = line.split(csvSplitBy);
                int contador = 0;
                for(int k=0; k<730; k=k+56){
                    for(int i=k; i<28+k; i=i+2){
                        if(contador!=0){
                            sb.append(",");
                        }
                        int value = Integer.valueOf(image[i]) +
                            Integer.valueOf(image[i+1]) +
                            Integer.valueOf(image[i+28]) +
                            Integer.valueOf(image[i+29]);

                        value = value/4;
                        sb.append(value);
                        contador++;
                    }
                }
                sb.append('\n');
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        pw.write(sb.toString());
        pw.close();
    }
}

```

Como último proceso antes de la selección del algoritmo de Data Minig, el archivo CSV generado por la aplicación Java, debió ser convertido a un archivo tipo ARFF que es el formato aceptado por Weka.

Para realizar esta conversión, se cambió manualmente la extensión del archivo y se agregaron los siguientes encabezados:

```

@attribute label {0,1,2,3,4,5,6,7,8,9}
@attribute pixel0 numeric
@attribute pixel1 numeric
@attribute pixel2 numeric
@attribute pixel3 numeric
@attribute pixel4 numeric
.

```

```
.  
.  
@attribute pixel195 numeric
```

## 2.4. Selección del algoritmo de minería de datos

Después de realizar algunas pruebas con Weka, el algoritmo que generó un mayor porcentaje de instancias correctamente clasificadas fue **Random Forest**.

Random Forests es una técnica de aprendizaje en conjunto. Es un híbrido del algoritmo de Bagging y el método de subespacio aleatorio, y utiliza árboles de decisión como clasificador de base. Cada árbol se construye a partir de una muestra de arranque del conjunto de datos original. Un punto importante es que los árboles no están sujetos a poda después de la construcción, lo que les permite estar parcialmente sobreajustados (overfitted) a su propia muestra de datos.

Para diversificar aún más los clasificadores, en cada rama del árbol, la decisión de qué característica dividir está restringida a un subconjunto aleatorio de tamaño  $n$ , del conjunto completo de características. El subconjunto aleatorio se elige de nuevo para cada punto de ramificación.  $n$  se sugiere que sea  $\log_2(N + 1)$ , donde  $N$  es el tamaño de todo el conjunto de características [8]. El término aparece de la primera propuesta de Random decision forests, hecha por Tin Kam Ho de Bell Labs en 1995.

Con este método se seleccionan aleatoriamente registros del conjunto de entrenamiento y se distribuyen en varios subconjuntos, sin que ningún registro se repita en dos o más conjuntos. Posteriormente se genera un árbol de decisión de cada subconjunto, formando así una gran colección de árboles de decisión, lo que le da el nombre de “forest” ya que se crea “un bosque” con los árboles generados.

Todos estos árboles de decisión se usan para crear un ranking de clasificadores. El algoritmo usa cada registro del conjunto de entrenamiento y utiliza cada uno de los árboles generados para obtener un valor de la clase, al final se selecciona por cada registro el valor con mayor número de “votaciones” entre el conjunto de árboles generados [9].

Este algoritmo era especialmente apropiado para esta competición, ya que además de correr rápido en bases de datos grandes, tiene la característica de que para conjuntos de datos con gran cantidad de registros, genera clasificadores muy certeros.

Una de las desventajas de este algoritmo es que, a diferencia de los árboles de decisión, la clasificación hecha por Random Forest es difícil de interpretar por el usuario.

## 2.5. Aplicación del algoritmo de minería de datos

Después de la reducción realizada, se probaron varios algoritmos de clasificación en Weka y se subieron los test a Kaggle obteniendo buenos resultados, pero para esta competición, la aplicación final del algoritmo de Data Mining se realizó con un

script codificado en Python y las siguientes tres librerías para el tratamiento de datos.

### **numpy**

Es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel. El ancestro de NumPy, Numeric, fue creado originalmente por Jim Hugunin con algunas contribuciones de otros desarrolladores. En 2005, Travis Oliphant creó NumPy incorporando características de Numarray en NumPy con algunas modificaciones. NumPy es open source [10].

En este ejercicio, esta extensión es utilizada para la generación del archivo final que se envía a Kaggle.

### **pandas**

pandas es una biblioteca de código abierto con licencia de BSD que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento y fácil de usar para el lenguaje de programación Python.

Una de las características de esta librería utilizadas en esta competición es el objeto DataFrame para la manipulación de datos con indexación integrada de los archivos CSV que se generaron en Java.

### **RandomForestClassifier**

Con esta librería se crea un metaesquemador que se adapta a una serie de clasificadores de árbol de decisiones en varias submuestras del conjunto de datos y utiliza un promedio para mejorar la precisión predictiva y controlar el ajuste excesivo. El tamaño de submuestra siempre es el mismo que el tamaño de muestra de entrada original [11]. Para la aplicación de esta librería, se utilizaron 2 parámetros:

- El primero es “**n\_estimators**” que permite configurar el número de árboles que se van a generar en el “bosque”, que en este caso fueron 1000
- El segundo es el número de núcleos del procesador que utilizará la librería en paralelo para ejecutar el algoritmo. El nombre del parámetro es “**n\_jobs**” y se le asignó el número 2.

El script utilizado fue el siguiente:

```
1. from sklearn.ensemble import RandomForestClassifier
2. import numpy as np
3. import pandas as pd
4.
5. dataset = pd.read_csv("train.csv")
6. target = dataset['label'].values.ravel()
7. train = dataset.iloc[:,1:].values
8.
```

```

9. test = pd.read_csv("test.csv").values
10.
11. rf = RandomForestClassifier(n_estimators=1000, n_jobs=2)
12. rf.fit(train, target)
13. pred = rf.predict(test)
14.
15. np.savetxt('submission_rand_forest.csv',
np.c_[range(1,len(test)+1),pred], delimiter=',', header = 'ImageId,Label',
comments = '', fmt='%d')

```

En la línea 5 se crea una referencia al conjunto de datos de entrenamiento y después en la línea 6 se extraen los valores de cada dígito (los valores de clase). Luego se toman los valores de los píxeles excluyendo los valores de respuesta. Posteriormente en la línea 9 se realiza el mismo proceso con el conjunto de datos test.

En las líneas 11, 12 y 13 se utilizan los valores extraídos de los conjuntos de datos como parámetros de la función de la librería que implementa el algoritmo Random Forest.

Por último, en la línea 15, de una manera mucho más simple comparada con el proceso que se hace en Weka, se genera el archivo de respuestas con el formato exacto que requiere Kaggle para analizar los resultados.

## 2.6. Interpretación y envío a Kaggle

Como se mencionó previamente, antes de la aplicación del algoritmo de Data Mining usando la librería de Python se realizaron algunas pruebas con Weka. El proceso completo se describe a continuación.

### 2.6.1. Aplicación del algoritmo RandomTree en Weka

Usando una configuración de Cross-validation con un número de Folds 10, se obtuvo en el conjunto de datos de entrenamiento un porcentaje de instancias correctamente clasificadas de un 80.2% como se ve en la siguiente imagen:

Correctly Classified Instances	33709	80.2595 %
Incorrectly Classified Instances	8291	19.7405 %
Kappa statistic	0.7806	
Mean absolute error	0.0395	
Root mean squared error	0.1987	
Relative absolute error	21.9409 %	
Root relative squared error	66.2433 %	
Total Number of Instances	42000	

La matriz de confusión fue la siguiente, en la que puede verse que el dígito “1” fue el número con mayor cantidad de aciertos (4409) y el número con menor número de aciertos fue el 8, con un total de 2941. Nótese también que entre el número “3” y “5” el algoritmo tuvo una tasa de error importante, ya que 327 imágenes que


correspondían a “3” las clasificó como “5”, y 313 que eran “5” las clasificó como “3”.


```

a    b    c    d    e    f    g    h    i    j    <-- classified as
3660  7    61   49   29   89   94   25   76   42 |    a = 0
 2 4409  45   32   22   27   24   43   52   28 |    b = 1
80   32 3291 188   84   65   97  105  188   47 |    c = 2
55   40  188 3192  52  327   34   95  246  122 |    d = 3
34   23   72   37 3174  63   90  114   91  374 |    e = 4
76   35   55  313   71 2766 128   54  198   99 |    f = 5
92   30   79   38  113  126 3528  19   86   26 |    g = 6
21   49  112   72  135   44   14 3603  67  284 |    h = 7
75   58  165  270  117  196   61  59 2941  121 |    i = 8
30   20   55   88  335   85   25  290  115 3145 |    j = 9

```

Posteriormente, el modelo construido en Weka fue aplicado al conjunto de datos “test”<sup>1</sup> y se obtuvo una puntuación en Kaggle de 81.6%.

1397
new
Bryan Valencia

0.96814

Your Best Entry 

Your submission scored 0.81600, which is not an improvement of your best score. Keep trying!

## 2.6.2. Aplicación del algoritmo RandomForest en Weka

Para este algoritmo, también se usó una configuración de Cross-validation con Folds 10, obteniendo en el conjunto de entrenamiento un porcentaje de instancias correctamente clasificadas de 96%, que es una gran mejora teniendo en cuenta los resultados obtenidos con RandomTree.

```

Correctly Classified Instances      40424      96.2476 %
Incorrectly Classified Instances    1576       3.7524 %
Kappa statistic                    0.9583
Mean absolute error                 0.0436
Root mean squared error            0.1088
Relative absolute error            24.2197 %
Root relative squared error        36.2561 %
Total Number of Instances         42000

```

En la matriz de confusión, puede verse claramente como hay un número mucho más grande de instancias correctamente clasificadas por cada posible dígito de la clase, aunque sigue conservándose la tendencia de que el algoritmo está confundiendo dígitos “5” con dígitos “3”, ya que el mayor número de instancias incorrectamente fueron 87 dígitos “5” que fueron clasificados mal como dígitos “3”.


<sup>1</sup> El proceso para la aplicación de un modelo previamente entrenada en un nuevo conjunto de datos se explica en la sección “Interpretación y envío a Kaggle” de la competición del Titanic.


```

a    b    c    d    e    f    g    h    i    j    <-- classified as
4071  0    2    3    4    3    22   0    24   3 |    a = 0
0 4617  21   11   8    2    8    6    7    4 |    b = 1
17   8 4045  19   12   0    5   34   35   2 |    c = 2
10   7   61 4112   3   47   6   26   53  26 |    d = 3
3    9   13   1 3919   1   16   8   12  90 |    e = 4
17   3    5   83  10 3590  41   3   25  18 |    f = 5
23   6    1    0   10  31 4050   1   15   0 |    g = 6
4   20  43   2   27   0    0 4217  10  78 |    h = 7
9   18  18  52  22  27  10   8 3865  34 |    i = 8
23   9    4  72  62   8    3   39  30 3938 |    j = 9

```

Posteriormente, el modelo construido en Weka fue aplicado al conjunto de datos “test” y se obtuvo una puntuación en Kaggle de 96.3%.

1397
new
Bryan Valencia

0.96814

Your Best Entry 

Your submission scored 0.96342, which is not an improvement of your best score. Keep trying!

### 2.6.3. Aplicación del algoritmo RandomForest en Python

La aplicación del algoritmo en Python pudo realizarse con el conjunto de características completas, es decir, con 784 píxeles por cada imagen, ya que esta herramienta aprovecha y maneja mejor los recursos de la máquina pudiendo generar resultados en cuestión de 2 minutos.

Como se explicó en la sección anterior, con la librería numpy, no es necesario realizar ninguna preparación del conjunto de datos para enviar a Kaggle ya que estos se generan exactamente como se requieren en la competición, esto quiere decir un archivo CSV con dos columnas, la primera indicando el ID de la imagen que es un número incremental del 1 al 28000 y la siguiente columna con el dígito que el modelo considera pertenece a ese conjunto de píxeles, así:

```

ImageId,Label
1,2
2,0
3,9
4,9
5,3
.
.
.
28000,2

```


La publicación de este archivo de “test” en Kaggle mostró una mejoría del 0.4% respecto al resultado obtenido utilizando Weka.




1410

new

Bryan Valencia




0.96814

Your Best Entry 

You advanced 91 places on the leaderboard!

Your submission scored 0.96814, which is an improvement of your previous score of 0.96628. Great job!

 Tweet this!

## Conclusiones

En literatura relacionada a los métodos para llevar a cabo procesos de KDD, es muy común encontrarse con que las fases de preprocesamiento pueden llevar hasta el 80% de todo el trabajo en un proyecto de este tipo. En efecto, la realización de estos dos proyectos, deja como primera lección aprendida que gran parte del tiempo que se requiere para descubrir conocimiento en conjuntos de datos se utiliza limpiando y preparando la información.

Además de ser una etapa extensa, de ésta puede depender significativamente la exactitud y rendimiento del algoritmo de Data Mining, ya que de una correcta estrategia de Feature Selection y/o Extraction pueden removerse características que no le aportan valor al proceso y que pueden hacerlo más lento, como también pueden crearse algunos atributos que permiten clasificar mejor el conjunto de datos.

Asimismo, también puede concluirse que el conocimiento del dominio del problema fue fundamental para llevar cabo estas actividades. Sin conocer el contexto de los datos, se hace difícil generar nuevas características o eliminar algunas que no representan utilidad para el minero de datos.

Sólo por mencionar un ejemplo, la división del campo “cabin” entre cabina y cubierta en la competencia del Titanic, sólo fue posible llevarla a cabo al tener suficiente conocimiento del dominio para entender que en este barco habían cubiertas nombradas con las letras de la “A” a la “G” y que por su ubicación éstas podían tener relación con la supervivencia de los pasajeros. Sin ese conocimiento del dominio, esa división no habría podido ser realizada.

El entendimiento del modelo generado por el algoritmo de Data Mining es otro elemento de vital importancia para el minero de datos. Esto fue notorio en la competencia del Titanic, ya que al aplicar algoritmos de reglas y árbol donde era posible ver qué atributos éstos estaban utilizando del conjunto de datos para realizar la clasificación, fue posible priorizarlos y desarrollar estrategias basados en aquellas características que contenían datos no homogéneos o vacíos para darle mejores herramientas al algoritmo para la construcción del modelo.

Específicamente, puede mencionarse que la decisión de generar un atributo de tarifa por persona nació al notar que la característica “fare” estaba siendo tenida en cuenta por los algoritmos. Al contrario en algoritmos como el Random Forest o basados en redes neuronales, donde el minero de datos no tiene la posibilidad de ver o entender el modelo construido, este tipo de ventajas se pierden.

En la competición para el reconocimiento de números, pudo comprobarse la efectividad del algoritmo Random Forest en conjuntos de datos grandes. Fue el algoritmo que mayor porcentaje de instancias correctamente clasificadas generó, lo que permite concluir que este algoritmo es, en efecto, una buena alternativa para conjuntos de datos con estas características.

Por otra parte, la realización de este proyecto demostró que Kaggle representa una muy buena oportunidad no sólo para aquellas personas con conocimiento y experiencia en Data Science mediante las competiciones con premios en dinero, sino también para aquellos que desean iniciar su experiencia en esta área, ya que en esta plataforma, además de encontrar los conjuntos de datos en sí, es posible seguir varios tutoriales, herramientas de ayuda e incluso participar en foros donde la comunidad alrededor del tema puede ayudarse.

## Referencias

- [1] Y. Jiang, L. Wang y L. Chen, «Discovering Interesting Classification Rules with Particle Swarm Algorithm,» de *Asia-Pacific Web Conference*, Berlin, 2008.
- [2] «Kaggle,» [En línea]. Available: <https://www.kaggle.com/>. [Último acceso: 12 Enero 2018].
- [3] «Weka 3: Data Mining Software in Java,» [En línea]. Available: <https://www.cs.waikato.ac.nz/ml/weka/>. [Último acceso: 12 Enero 2018].
- [4] U. Fayyad, G. Piatetsky-Shapiro y P. Smyth, «From Data Mining to Knowledge Discovery in Databases,» *AI Magazine*, vol. 17, nº 3, pp. 37-54, 1996.
- [5] «Encyclopedia Titanica,» [En línea]. Available: <https://www.encyclopedia-titanica.org/>. [Último acceso: 12 Enero 2018].
- [6] «Titanic Cross Section Diagram,» 25 Junio 2014. [En línea]. Available: <https://openclipart.org/detail/194470/titanic-cross-section-diagram>. [Último acceso: 11 Enero 2018].
- [7] J. Espino y J. Tijerina, «Algoritmo C4.5,» 2005. [En línea]. Available: [www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas\\_alumnos/C4.5/C4.5\(2005-II-B\).pdf](http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/C4.5/C4.5(2005-II-B).pdf). [Último acceso: 12 Enero 2018].
- [8] C. Sammut y G. Webb, «Random Forests,» de *Encyclopedia of Machine Learning and Data Mining*, Boston, Springer, 2017, pp. 1054-1054.
- [9] T. Sehn, «How Random Forest algorithm works,» 2014. [En línea]. Available: <https://www.youtube.com/watch?v=loNcrMjYh64>. [Último acceso: 11 Enero 2018].
- [10] «NumPy,» 2017. [En línea]. Available: <http://www.numpy.org/>. [Último acceso: 12 Enero 2018].
- [11] «RandomForestClassifier,» [En línea]. Available: <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. [Último acceso: 12 Enero 2018].