UNIVERSIDAD DE SEVILLA

Ciencia del dato aplicada: Competiciones en Kaggle

Trabajo Fin de Grado - Grado en Estadística

Departamento de Ciencias de la computación e Inteligencia Artificial

Facultad de Matemáticas

Curso Académico 2015/16

TRABAJO REALIZADO POR:

ANTONIO JOSÉ BARRIOS MARÍN

TUTOR:

ÁLVARO ROMERO JIMÉNEZ

Índice general

Resumen **3**

Abstract **5**

Capítulo 1: Introducción al Aprendizaje Automático **7**

Capítulo 2: Introducción a *Kaggle* **9**

Capítulo 3: Métodos de clasificación **11**

Análisis discriminante lineal 11

Árboles de clasificación 13

Random forests 16

Capítulo 4: Competiciones **21**

Competición Titanic 21

Competición animales de acogida (*shelter animals*) 39

Capítulo 5: Herramientas utilizadas **53**

Paquetes de *R* utilizados 53

Conclusiones **57**

Bibliografía **59**

Resumen

El objetivo de este trabajo es la aplicación de algoritmos de aprendizaje automático para resolver problemas de modelización predictiva planteados en la plataforma *Kaggle*.

El trabajo comienza con una introducción a lo que se entiende por aprendizaje automático, describiéndose los diversos tipos de algoritmos aplicables y los diferentes tipos de aprendizaje existentes. Además, se explica en que consiste la plataforma web *Kaggle*, detallando su funcionamiento así como los diferentes tipos de competiciones que incluye.

Tras describir brevemente algunos algoritmos de aprendizaje automático que se aplicarán posteriormente en el trabajo, luego se desarrollará el núcleo del mismo: la participación en dos competiciones de *Kaggle*, de diferentes rangos de dificultad cada una, enfrentándonos de esta manera al tratamiento y utilización de conjuntos de datos 'reales' para el desarrollo de un modelo predictivo.

Abstract

The aim of this work is Application of machine learning algorithms to solve problems raised in predictive modeling platform *Kaggle*.

This Work begins with an introduction to what is meant by Machine Learning, describing various types of algorithms applicable and different Ttypes of learning existing. In addition, it explains what the Web platform *Kaggle*, detailing how it´s work and the Different Types of competitions that are including on the web.

After describing briefly some algorithms machine learning that subsequently apply at this work, then it will develop the core of: the Participation in two competitions *Kaggle* of different ranges of difficulty each one, facing the treatment and use of 'real' data sets for the development of a predictive model.

Capítulo 1: Introducción al Aprendizaje Automático

El Aprendizaje Automático es la rama de la Inteligencia Artificial que tiene como objetivo desarrollar técnicas que permitan a los ordenadores aprender. De forma más concreta, se trata de crear algoritmos capaces de generalizar comportamientos y reconocer patrones a partir de una información suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento, es decir, un método que permite obtener por generalización un enunciado general a partir de enunciados que describen casos particulares.

A un nivel muy básico, podríamos decir que una de las tareas del aprendizaje automático es intentar extraer conocimiento sobre algunas propiedades no observadas de un objeto basándose en las propiedades que sí han sido observadas de ese mismo objeto (o incluso de propiedades observadas en otros objetos similares), es decir, predecir el comportamiento futuro a partir de lo que ha ocurrido en el pasado.

Hay un gran número de métodos que se engloban dentro de lo que llamamos aprendizaje inductivo. La principal diferencia entre ellos radica en el resultado predicho que proporcionan. Algunos tipos habituales de algoritmos de aprendizaje son:

- *Regresión*: predicen un valor real. Por ejemplo, predecir el valor de la bolsa a partir de su comportamiento anterior.
- *Clasificación*: predicen la clasificación de objetos sobre un conjunto de clases prefijadas. Si solo se permiten dos posibles clases, entonces tenemos lo que se llama clasificación *binaria*. Si se permiten más de dos clases, estamos hablando de clasificación *multiclase*. Por ejemplo, clasificar si una determinada noticia es de deportes, entretenimiento, política, etc.
- *Ranking*: predicen el orden óptimo de un conjunto de objetos según un orden de relevancia predefinido. Por ejemplo, el orden en que un buscador devuelve recursos de Internet como respuesta a una búsqueda de un usuario.

Normalmente, cuando se aborda un nuevo problema de aprendizaje automático lo primero que se hace es enmarcarlo dentro de alguno de los tipos anteriores, ya que de ello depende la medida del error cometido entre la predicción y la realidad. En consecuencia, el problema de medir cómo de acertado es el aprendizaje obtenido deberá ser tratado para cada caso particular de metodología aplicada, aunque en general podemos adelantar que necesitaremos

"embeber" la representación del problema en un espacio en el que tengamos definida una medida.

Dependiendo del tipo de salida que se produzca y de cómo se aborde el tratamiento de los ejemplos, los diferentes algoritmos de aprendizaje automático se pueden catalogar como sigue:

- Aprendizaje *supervisado*: se genera una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema, donde la base de conocimientos del sistema está formada por ejemplos etiquetados a priori (es decir, ejemplos de los que sabemos su clasificación correcta).
- Aprendizaje *no supervisado*: la modelización se lleva a cabo a partir de un conjunto de ejemplos formados únicamente por entradas al sistema, sin conocer su clasificación correcta. Se busca que el sistema sea capaz de reconocer patrones para poder etiquetar las nuevas entradas.
- Aprendizaje *semisupervisado*: es una combinación de los dos tipos de aprendizaje anteriores, a partir de ejemplos cuya clasificación es conocida y ejemplos cuya clasificación no es conocida.
- Aprendizaje *por refuerzo*: los algoritmos aprenden observando el mundo que le rodea, con un continuo flujo de información en las dos direcciones (del mundo a la máquina y de la máquina al mundo) realizando un proceso de ensayo y error, y reforzando aquellas acciones que reciben una respuesta positiva en el mundo.
- *Transducción*: es similar al aprendizaje supervisado, pero su objetivo no es construir de forma explícita una función, sino únicamente tratar de predecir las categorías a las que pertenecen los siguientes ejemplos basándose en los ejemplos de entrada, sus respectivas categorías y los ejemplos nuevos introducidos al sistema.
- Aprendizaje *multitarea*: engloba todos aquellos métodos de aprendizaje que usan conocimiento aprendido previamente por el sistema a la hora de enfrentarse a problemas parecidos a los ya afrontados anteriormente.

Capítulo 2: Introducción a *Kaggle*

Kaggle es una plataforma de competiciones de análisis y modelización predictiva de datos proporcionados por empresas e investigadores. Fue fundada en 2010 por el economista australiano Anthony Goldbloom, inspirándose en la competición organizada por la plataforma Netflix de vídeo bajo demanda para mejorar su software de recomendación de títulos. A día de hoy han tenido lugar en *Kaggle* más de 200 competiciones, con premios de hasta 3 millones de dólares.

El funcionamiento de la plataforma es muy sencillo: un promotor contacta con el equipo de *Kaggle* y prepara un conjunto de datos de su negocio o investigación. Una parte de estos datos son publicados en la web para que los concursantes desarrollen sus modelos predictivos, mientras que la otra parte se reserva de manera privada para permitir la clasificación de los concursantes en función del comportamiento de sus modelos y, por tanto, determinar el ganador de la competición. Los datos públicos contienen la variable respuesta que se necesita modelizar (se trata pues de aprendizaje *supervisado*), y para valorar la bondad de las predicciones se publica también una métrica, es decir, una fórmula para medir el error. Finalmente, se comparan las predicciones sobre los datos privados, obteniendo así el poder predictivo de los modelos y, por tanto, estableciendo una tabla de posiciones en función del valor obtenido. Realmente hay una tabla de posiciones pública, en la que los modelos se clasifican sobre una parte de los datos privados a medida que se van subiendo a la plataforma, y una tabla de posiciones privada, que se calcula una vez alcanzado el plazo límite de la competición, usando una parte de los datos privados reservada al efecto, y que determina la posición final alcanzada por cada concursante.

Existen varios tipos de competiciones:

- Competiciones *Getting Started*: son aquellas competiciones que proporciona el propio *Kaggle* para que los usuarios se introduzcan al campo del aprendizaje automático con conjuntos de datos de fácil tratamiento. Por ejemplo, la determinación de la supervivencia de los pasajeros y la tripulación del Titanic.
- Competiciones *Playground*: al igual que las competiciones *Getting Started*, sirven de práctica para los usuarios. Por ejemplo, la clasificación del tipo de crimen en la ciudad de San Francisco.
- Competiciones *Featured Datasets*: conjuntos de datos destacados que presentan un nivel de dificultad superior a los dos mencionados anteriormente. Por ejemplo, la clasificación de los salarios de las personas de la ciudad de San Francisco.

- Competiciones *Recruitment*: competiciones a través de las cuales los usuarios que mejores resultados obtienen con sus modelos tienen la oportunidad de encontrar un trabajo. Por ejemplo, la clasificación por etiquetas de atributos para restaurantes usando fotografías de usuarios registrados en la web *Yelp*.
- Competiciones *Featured*: son competiciones del más alto nivel de dificultad, como las competiciones *Recruitment*, pero cuyo premio no es una oferta de trabajo, sino una compensación monetaria. Por ejemplo, una clasificación de los clientes del Banco Santander para ver qué clientes están contentos con el banco, donde se puede obtener un premio de 60.000 dólares.

Capítulo 3: Métodos de clasificación

A continuación, se describen con detalle los métodos de aprendizaje supervisado que se han aplicado en este trabajo: análisis discriminante lineal, árboles de decisión y selvas aleatorias (*random forests*).

Análisis discriminante lineal

El análisis discriminante lineal (*linear discriminant analysis*, LDA) es una generalización del discriminante lineal de Fisher, un método utilizado en estadística, reconocimiento de patrones y aprendizaje automático para encontrar una combinación lineal de rasgos que caracterizan o separan dos o más clases de objetos. La combinación resultante puede ser utilizada para una reducción de dimensiones, o, como es en nuestro caso, para un clasificador lineal.

Para ilustrar cómo el LDA se aplicaría a un problema de clasificación binaria, considérese un conjunto de entrenamiento en el que cada caso viene dado por los valores de un conjunto de características $\mathbf{x}$ continuas y de los que se conoce si pertenecen a la clase 0 o a la clase 1.

LDA asume que las probabilidades condicionales $P(\mathbf{x}|Y=0)$ y $P(\mathbf{x}|Y=1)$ siguen una distribución normal multivariante de media $\mu_\mathbf{0}$ y $\mu_\mathbf{1}$, respectivamente, y de matriz de covarianzas Σ de rango completo e idéntica para ambas (asunción de homocedasticidad).

Bajo estas condiciones y siguiendo un enfoque bayesiano, operando convenientemente se llega a que,

$$P(Y=1|\mathbf{x}) = \frac{1}{1+\mathrm{e}^{\mathbf{w}^T\cdot(\mathbf{x}-\mathbf{c})}}$$

donde $\mathbf{w} = \Sigma^{-1}\cdot(\mu_\mathbf{0}-\mu_\mathbf{0})$ y $\mathbf{c}$ es una constante que depende de $\mu_\mathbf{0}$, $\mu_\mathbf{1}$ y de las probabilidades a priori π_0 y π_1 de pertenencia a las clases.

La regla de clasificación tiene, por tanto, una interpretación geométrica: desplazar $\mathbf{x}$ por $\mathbf{c}$, proyectar sobre la dirección $\mathbf{w}$ y determinar si el resultado es positivo, en cuyo caso clasificamos $\mathbf{x}$ en la clase 0, o negativo, en cuyo caso clasificamos $\mathbf{x}$ en la clase 1.

En el caso de que $\Sigma = \sigma\cdot I$, es decir, las características son independientes, entonces $\mathbf{w}$ está en la dirección $\mu_\mathbf{1}-\mu_\mathbf{0}$. Por tanto, $\mathbf{x}$ se clasifica en función de si la proyección sobre $\mathbf{w}$ está más cerca (según la distancia euclídea) de $\mu_\mathbf{0}$ o de $\mu_\mathbf{1}$. Además, si $\pi_0 = \pi_1$, entonces

$\mathbf{c} = (\mu_\mathbf{1} + \mu_\mathbf{0})/2$, el punto medio entre las dos medias. Si $\pi_1 > \pi_0$, entonces $\mathbf{c}$ queda más cerca de $\mu_\mathbf{0}$, por lo que una mayor parte de la línea pertenece a la clase 1 a priori. A la inversa, si $\pi_0 > \pi_1$, entonces la frontera se desplaza hacia $\mu_\mathbf{1}$.

El procedimiento anterior se puede generalizar cuando el número de clases es mayor que dos. En este caso,

$$\ln(P(Y = i|\mathbf{x})) \propto \mathbf{w}_i^T \times \mathbf{x} + c_i \tag{1}$$

donde $\mathbf{w}_i = \Sigma^{-1} \times \mu_i$ y c_i es una constante que depende de $\mu_\mathbf{i}$ y de la probabilidad a priori de pertenencia a la clase i. La interpretación geométrica es clasificar $\mathbf{x}$ en la clase cuyo centroide $\mu_\mathbf{i}$ quede más cercano, según la distancia de Mahalanobis.

Al entrenar el modelo en un conjunto de datos lo que se hace principalmente es calcular los vectores de media de cada grupo de la variable dependiente y la matriz de varianzas y covarianzas Σ, a través de métodos como el de los momentos o el método de máxima verosimilitud entre otros.

Análisis discriminante lineal en R

La función `lda` del paquete `MASS` implementa el análisis discriminante lineal. La manera más habitual de aplicarla es

```
lda(Y ~ X1 + X2 + ... + Xn, data)
```

donde `Y` indica la variable dependiente (cuyo tipo de datos debe ser `factor`), `X1`,`X2`,...,`Xn` indican las variables predictoras (cuyo tipo de datos debe ser `numeric`) y `data` es un marco de datos con los valores de esas variables en el conjunto de entrenamiento.

El resultado obtenido es un objeto de clase `lda` que contiene entre, otras componentes, las probabilidades a priori de pertenencia a las clases y las medias de los grupos, así como una matriz con los pesos de las funciones de discriminación.

El paquete también implementa un método para la función genérica `predict`, que le permite manejar objetos de clase `lda`. De esta manera, es posible clasificar nuevos casos mediante la expresión

```
predict(object, newdata)
```

donde `object` es un objeto de clase `lda` y `newdata` es un marco de datos con los valores de las características de los nuevos casos. El resultado es una lista que contiene una componente `class` con la clasificación predicha de cada nuevo caso y una componente `posterior` con las probabilidades a posteriori de pertenencia a cada clase para cada nuevo caso.

A la hora de aplicar tanto la función `lda` como la función `predict`, se pueden establecer las probabilidades a priori de pertenencia a cada una de las clases mediante el argumento `prior`. Por defecto se toman las proporciones en el conjunto de entrenamiento.

Árboles de clasificación

Los árboles de clasificación y de regresión (*Classification and regression trees*, CARTs) son métodos de aprendizaje automático para la construcción de modelos de predicción a partir de conjuntos de datos. Estos modelos se obtienen particionando recursivamente el espacio de datos y ajustando un modelo de predicción simple dentro de cada partición.

Los modelos obtenidos se pueden representar como árboles de decisión, en las que las hojas simbolizan etiquetas de clase y las ramas conjunciones de características que llevan a esas etiquetas de clase. Los árboles de decisión obtenidos para variables dependientes que toman un número finito de valores se denominan árboles de clasificación, mientras que los obtenidos para variables dependientes continuas se denominan árboles de regresión. En este trabajo solo se usarán los primeros.

Existen muchos algoritmos de entrenamiento de árboles de clasificación. La mayoría de ellos construye un árbol a partir del siguiente esquema:

1. Si el conjunto de datos D es suficientemente homogéneo, entonces devolver la etiqueta de clase más apropiada para D.
2. En caso contrario, determinar la mejor partición S de D.
3. Para cada subconjunto D_i de D dado por S, construir un árbol T_i de manera recursiva a partir de D_i.
4. Devolver el árbol cuya raíz está etiquetada por S y cuyos hijos son los árboles T_i.

En general, se considera que un conjunto de instancias es homogéneo si la mayoría de ellas pertenecen a la misma clase, y la etiqueta de clase más apropiada es esa clase mayoritaria.

La determinación de la mejor partición se puede hacer de muchas formas. Por ejemplo, el algoritmo *CART* (desarrollado por Leo Breiman y otros en 1984) considera particiones binarias de la forma $X \in S$ y $X \notin S$, para X una característica categórica y S un subconjunto de posibles valores de X, y de la forma, $X \leq c$ y $X > c$ para X una variable continua. Entonces se elige la partición que proporciona una mayor reducción de la devianza (una medida del ajuste del modelo probabilístico representado por el árbol) o del índice de Gini (una medida de la impureza de los subconjuntos inducidos por la partición representada por el árbol).

Los árboles de clasificación presentan como principales ventajas que son capaces de manejar características tanto discretas como continuas y que ofrecen modelos de caja blanca, en los que es fácil entender e interpretar las respuestas proporcionadas. Por contra, está demostrado que el aprendizaje de un árbol de clasificación óptimo para un conjunto de datos es un problema del que en principio no se puede esperar que se pueda resolver de manera eficiente (esto es un problema NP-completo). Es por ello que, en la práctica, los algoritmos de aprendizaje proporcionan árboles de clasificación sobreajustados a los datos, lo que hace necesario realizar un proceso posterior de poda que simplifique el árbol construido, permitiendo de esta forma una mejor generalización del conjunto de datos.

Árboles de clasificación en R

La función `tree` del paquete `tree` implementa el algoritmo *CART* para la construcción de árboles de clasificación y para aplicar dicha función necesitamos cargar el paquete `tree` mediante . La manera habitual de aplicarla es

```
tree(Y ~ X1+ X2 + ... +Xn , data)
```

donde `Y` indica la variable dependiente, `X1`, `X2`,..., `Xn` indican las variables predictoras y `data` es un marco de datos con los valores de esas variables en el conjunto de entrenamiento. Si la variable dependiente es de tipo `factor` se construirá un árbol de clasificación. Mientras que si es de tipo `numeric` se construirá un árbol de regresión. Con el argumento `split`, que puede tomar como valor la cadena `"deviance"` (valor por defecto) o la cadena `"Gini"`, se establece el mecanismo para determinar la mejor partición de conjunto de datos.

El resultado proporcionado por la función `tree` es un objeto de clase `tree` que contiene, entre otras componentes, un marco de datos con una fila por cada nodo del árbol que guarda toda la información relativa a ese nodo y un vector que asocia cada instancia del conjunto de entrenamiento con el nodo del árbol que la clasifica.

Siendo un ejemplo de como se representa un árbol de clasificación:

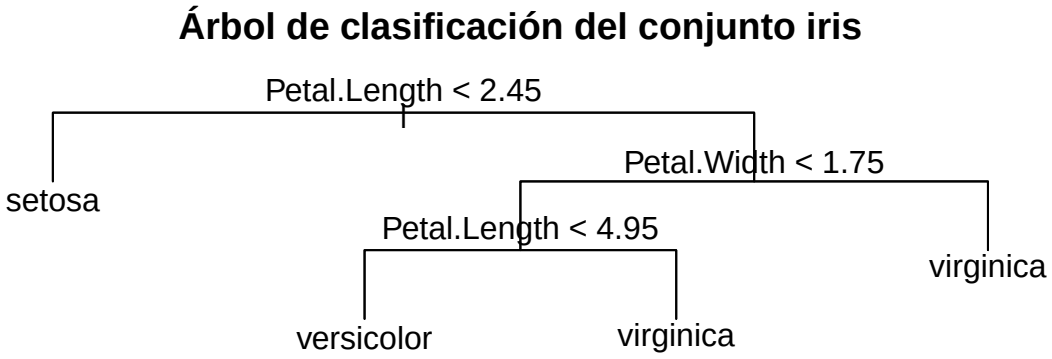

Figura 1: Ejemplo representando un árbol de clasificación

La figura 1 muestra un árbol de clasificación construido a partir del conocido conjunto de datos *iris*, que contiene varias características de distintas flores de tres especies de lirios. Según este árbol, una flor con longitud de pétalo menor que 2.45 sería clasificada como de la especie *setosa*. Por el contrario, una flor con longitud de pétalo mayor que 4.95 y anchura de pétalo menor que 1.75 sería clasificada como de la especie *virginica*.

El paquete también implementa un método para la función genérica `predict`, que le permite manejar objetos de la clase `tree`. De esta manera, es posible clasificar nuevos casos mediante la expresión

```
predict(object, newdata, type = c("vector", "class"))
```

donde `object` es un objeto de clase `tree` y `newdata` es un marco de datos con los valores de las características de los nuevos casos. El resultado obtenido depende del valor del argumento `type`: para la cadena `"vector"` (valor por defecto) es una matriz con las probabilidades de pertenencia a cada clase para cada caso de `newdata`; para la cadena `class` es un factor con la clase más probable para cada caso de `newdata`, deshaciendo los empates de manera aleatoria.

Random forests

Los métodos de agregación de modelos son una de las técnicas más potentes en aprendizaje automático. Aunque abarcan una gran diversidad de procedimientos, en esencia se pueden reducir a dos ideas fundamentales:

1. Construir múltiples y diversos modelos predictivos a partir de versiones adaptadas del conjunto de entrenamiento (generalmente mediante muestreos aleatorios o ponderaciones).
2. Combinar las predicciones de estos modelos (generalmente mediante votación o promediado).

Las agregaciones de modelos suelen ofrecer un mayor rendimiento que los modelos individuales, ya que permiten reducir el sesgo (cómo de cerca de la función objetivo se encuentra en promedio el clasificador proporcionado por el algoritmo de aprendizaje) y la varianza (cómo de dependiente del conjunto de entrenamiento es el clasificador proporcionado por el algoritmo de aprendizaje). Como contrapartida, se incrementa la complejidad algorítmica y del modelo obtenido.

Una de las primeras técnicas de agregación de modelos que se propusieron (por *L. Breiman* en 1996) es *bagging*, abreviatura de *bootstrap aggregation*. La idea básica consiste en generar muestras aleatorias con reemplazamiento (*bootstrap samples*) del conjunto de entrenamiento y aprender un conjunto de clasificadores a partir de ellas. Para predecir la clase de una nueva instancia bastaría obtener las respuestas individuales de estos modelos y combinarlas de alguna manera, por ejemplo mediante votación (la clase predicha sería la clase mayoritaria) o mediante promediado (más adecuado si las respuestas de los modelos son probabilidades).

Cuando se aplica la técnica de *bagging* a los árboles de clasificación es importante notar que los distintos árboles generados no son completamente independientes entre sí. Esto es debido a que, a pesar de que cada uno de ellos se entrena a partir de un conjunto de datos distinto, todas las variables predictoras se consideran en todas las particiones de todos los árboles. Esta propiedad se conoce como correlación entre los árboles e impide que la técnica reduzca la varianza de manera óptima.

Para reducir esta correlación se propusieron varios mecanismos que introducían aleatoriedad en el proceso de entrenamiento y que fueron unificados por *L. Breiman* en 2001 en el modelo conocido como *random forest*. El algoritmo general para construir un modelo de este tipo es el siguiente:

1. Establecer la cantidad de modelos a construir.
2. Para cada uno de esos modelos:
    1. Generar una muestra *bootstrap* del conjunto de entrenamiento.
    2. Entrenar un árbol de clasificación a partir de esa muestra de tal manera que, para cada partición a realizar:
        1. Seleccionar aleatoriamente k variables predictoras.
        2. Elegir la mejor variable predictora de entre las seleccionadas.
    3. Terminar de construir el árbol cuando se cumpla algún criterio típico de parada, pero no podar el árbol.

Estimación de la importancia de las variables predictoras

La naturaleza agregadora del modelo *random forest* hace imposible obtener una comprensión de la relación entre las variables predictora y la variable dependiente. No obstante, es posible cuantificar el impacto de cada variable predictora.

Una primera estrategia consiste en comparar, para cada variable predictora, el error de predicción sobre las instancias *out-of-bag* con el obtenido al permutar aleatoriamente los valores de la variable en esas instancias. Promediando sobre todos los árboles del *random forest* y tipificando mediante la desviación típica, se calcula la importancia relativa de cada variable predictora.

Otra estrategia consiste en calcular, para cada variable predictora y promediando sobre todos los árboles, la disminución total de la impureza (medida con el índice de Gini) de los nodos en los que la variable ha inducido la partición del conjunto de datos.

Estimación *out-of-bag* del rendimiento predictivo

Una ventaja del modelo *random forest* (y en general, de los modelos obtenidos mediante la técnica de *bagging*) es que pueden proporcionar su propia estimación interna del rendimiento predictivo, estimación que correlaciona bien con las estimaciones obtenidas a partir de un conjunto de prueba o mediante el método de validación cruzada.

Para calcular esta estimación, basta tener en cuenta que cada modelo del agregado se construye a partir de una muestra aleatoria con reemplazamiento del conjunto de entrenamiento. Por lo tanto, para cada uno de esos modelos hay instancias, llamadas *out-of-bag*, que no se consideran y que pueden utilizarse para estimar el error de predicción cometido por el modelo. Promediando esos errores se obtiene entonces una estimación del error predictivo de la agregación de los modelos.

Analizando empíricamente el comportamiento del error *out-of-bag*, se observa en la práctica que a menudo la mejora del rendimiento predictivo decrece de manera exponencial con el número de árboles construidos. Un ejemplo lo podemos observar en la figura 2, donde también se muestra la evolución de los promedios de la fracción de instancias de cada una de las dos clases del ejemplo que se han clasificado erróneamente.

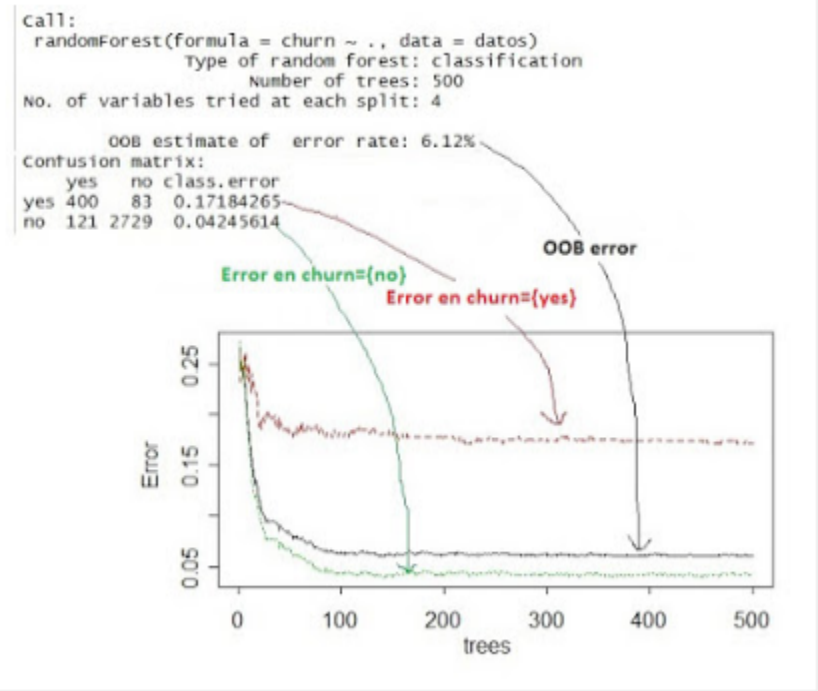

Figura 2: Representación gráfica del error producido en la clasificación de cada grupo de la variable dependiente del modelo *random forest* junto al error *out-of-bag*

Random forests en *R*

La función `randomForest` del paquete `randomForest` implementa el algoritmo de *Breiman* para la construcción de los *random forests*. La manera habitual de aplicar el algoritmo es:

```
randomForest(Y ~ X1 + X2 + ... + Xn, data)
```

donde `Y` indica la variable dependiente (que para un problema de clasificación debe ser un factor), `X1`, `X2`,..., `Xn` indican las variables predictoras y `data` es un marco de datos con los valores de esas variables. El argumento `ntree` establece la cantidad de árboles a agregar (por defecto 500) y el argumento `mtry` la cantidad de variables predictoras a elegir aleatoriamente en cada nodo (por defecto $\lfloor\sqrt{n}\rfloor$ para un problema de clasificación).

El resultado proporcionado por la función `randomForest` es un objeto de clase `randomForest` que contiene, entre otras componentes, una lista con los árboles construidos y una matriz con los promedios de los errores *OOB* y de clasificación errónea de cada clase.

El paquete también implementa un método para la función genérica `predict`, que le permite manejar los objetos de clase `randomForest`. De esta manera, es posible clasificar nuevas instancias mediante la expresión

```
predict(object, newdata, type = c("response", "prob", "votes"))
```

donde `object` es un objeto de clase `randomForest` y `newdata` es un marco de datos con los valores de las características de las nuevas instancias.

El resultado obtenido depende del valor del argumento `type`: para la cadena `response` (valor por defecto) es un factor con la clase mayoritaria predicha por árboles del *random forest* para cada instancia; para la cadena `"prob"` es una matriz con las probabilidades de pertenencia a cada clase para cada instancia; para la cadena `"vote"` es una matriz con los votos (frecuencias absolutas si el argumento `norm.votes` es `FALSE`, relativas en caso contrario) de pertenencia a cada clase para cada instancia.

El paquete también implementa un método para la función genérica `plot`, que le permite manejar objetos de clase `randomForest`, obteniendo una representación gráfica de la evolución del error *OOB* y de clasificación errónea en cada clase, a medida que se van construyendo los árboles, hasta la cantidad especificada.

Finalmente, el paquete también cuenta con la función `importance` para extraer de un objeto de clase `randomForest` la importancia relativa calculada de cada variable predictora. El argumento `type` de esta función controla la medida usada: reducción media del rendimiento, para el valor 1; reducción media de la impureza de los nodos, para el valor 2.

Capítulo 4: Competiciones

El núcleo del trabajo ha sido la participación en dos competiciones de *Kaggle*: la competición Titanic, una competición *Getting Started*, y la competición *shelter animals*, una competición de entrenamiento previo a las competiciones más difíciles de la plataforma.

Competición Titanic

El hundimiento del RMS Titanic es uno de los naufragios más funestos de la historia. El 15 de abril de 1912, durante su viaje inaugural, el Titanic se hundió después de chocar con un iceberg, matando a 1502 de 2224 personas, contando pasajeros y tripulación. Esta tragedia conmocionó a la comunidad internacional y condujo a mejores normas de seguridad aplicadas a los buques.

Una de las razones por las que el naufragio dió lugar a esa pérdida de vidas fue que no había suficientes botes salvavidas para los pasajerosy 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 aquellos de la clase alta.

En este desafío se pide que se apliquen las herramientas de aprendizaje automático para predecir qué pasajeros sobrevivieron a la tragedia.

La competición del Titanic es una competición de tipo *Getting Started* y proporciona un conjunto de datos de entrenamiento que consta de 891 ejemplos con 12 características:

- Supervivencia (0=No sobrevive, 1=Sobrevive)
- Clase del pasajero (Primera, Segunda, Tercera)
- Nombre
- Sexo
- Edad
- Sibsp (número de hermanos/as, marido o mujeres a bordo)
- Parch (números de padres/madres o hijos/as a bordo)
- Número de ticket
- Tarifa de ticket
- Cabina de alojamiento
- Puerto de embarque (C = Cherbourg, Q = Queenstown, S = Southampton)

Encontramos valores perdidos en numerosos casos en diversas variables. Para tratar esta dificultad en las variables numéricas, se pueden sustituir esos valores perdidos bien por el valor de la observación anterior, bien por la media de todas las observaciones. Por ejemplo, los valores perdidos de la variable `edad` se han imputado con el valor de la observación anterior, usando para ello la función `fill` del paquete `tidyr` de R.

Como características iniciales para construir los modelos hemos considerado las variables `edad`, `sexo` y `clase`. El modelo debe predecir la variable `supervivencia`.

Una vez que se ha decidido cuáles serán las variables predictoras que incluiremos en el modelo prodecemos a leer los conjuntos de datos de entrenamiento y de prueba.

```
titanicentrenamiento <- read.csv("traintitanic.csv",header=TRUE)

l <- list(supervivencia= titanicentrenamiento$Survived,
          edad= titanicentrenamiento$Age,
          clase= titanicentrenamiento$Pclass,
          sexo= titanicentrenamiento$Sex)
Titanicent<- as_data_frame(l) %>% fill(edad)

titanictest<-read.csv("testtitanic.csv",header=TRUE)
L <- list(edad= titanictest$Age,
          clase= titanictest$Pclass,
          sexo= titanictest$Sex)
Titanitest<- as_data_frame(L) %>% fill(edad)
```

Para comenzar, se ha realizado un análisis discriminante lineal, una técnica que en principio tiene poco sentido para este conjunto de datos, ya que este modelo de clasificación es adecuado para variables continuas. En nuestro caso, en cambio, aunque la variable `edad` sí es una variable numérica continua, las variables `sexo` y `clase` son dos variables categóricas. Puesto que el método no es en principio apropiado, no se esperan obtener buenos resultados en la predicción. Aun así se decidió seguir adelante para tener una primera toma de contacto con la competición y con *Kaggle* y para ver si modelos más adecuados proporcionaban mejores resultados.

```
ent.lda<- lda(supervivencia ~ edad + clase + sexo, data = Titanicent)
ent.lda
```

```
## Call:
## lda(supervivencia ~ edad + clase + sexo, data = Titanicent)
##
## Prior probabilities of groups:
##         0         1
## 0.6161616 0.3838384
##
```

```
## Group means:
##       edad    clase  sexomale
## 0 30.33439 2.531876 0.8524590
## 1 28.37307 1.950292 0.3187135
##
## Coefficients of linear discriminants:
##                   LD1
## edad     -0.01794016
## clase    -0.76295690
## sexomale -2.12999613
```

Calculamos ahora la predicción que realiza el modelo sobre cada ejemplo del conjunto de prueba.

```
prediccionlda<-predict(ent.lda,Titanitest)$class
head(prediccionlda,10)
```

```
##  [1] 0 1 0 0 1 0 1 0 1 0
## Levels: 0 1
```

En la figura 3 podemos ver una representación de la supervivencia predicha, por edades y sexo de los individuos.

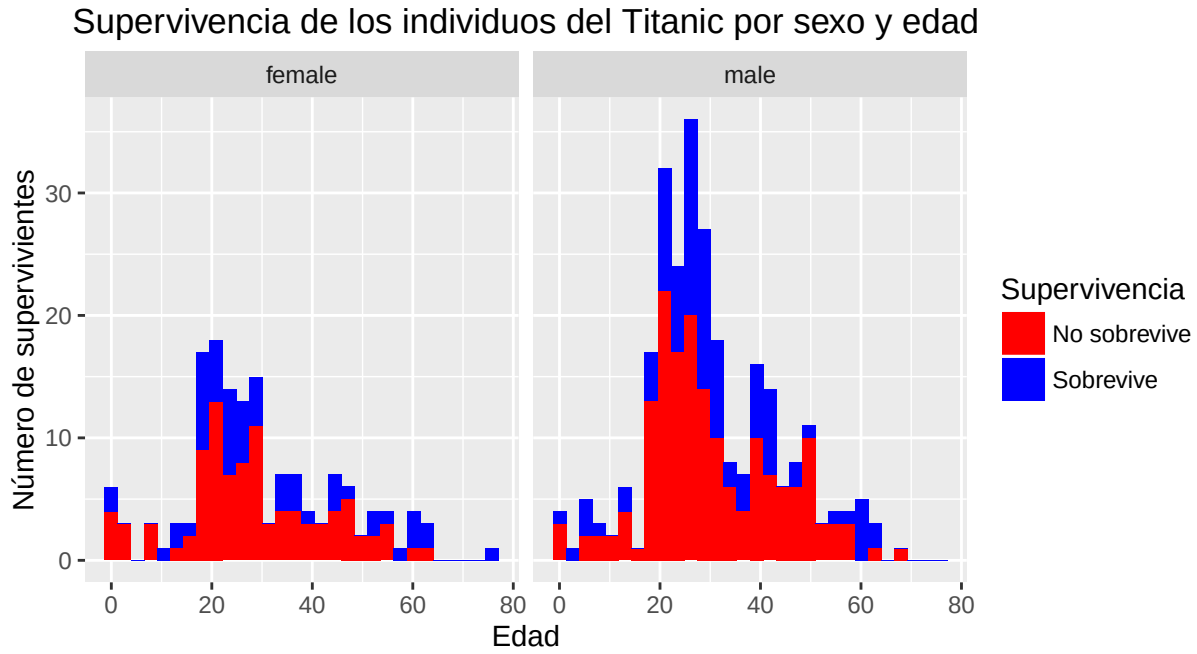

Figura 3: Representación de la supervivencia de los pasajeros del Titanic según las predicciones del modelo LDA

Finalmente, para valorar el comportamiento de nuestro modelo guardamos las predicciones obtenidas en un fichero con formato csv y lo subimos a la plataforma *Kaggle*.

```
solucionlda <- data.frame(PassengerID = titanictest$PassengerId,
                          Survived = prediccionlda)
write.csv(solucionlda, file = 'solucionlda.csv', row.names = F)
```

Obtenemos una puntuación de 0.77512 (es decir, las predicciones sobre el conjunto privado de datos son correctas en un 77.51 % de los casos),lo que nos sitúa en la posición 2513 de la tabla de posiciones pública(figura 4). Para conocer la posición final en la tabla de posiciones privada se debería esperar a que la competición termine. En este caso, el plazo termina el 31 de diciembre de 2016. No obstante, al ser una competición *Getting Started* no sabemos con certeza qué ocurrirá al llegar esa fecha.

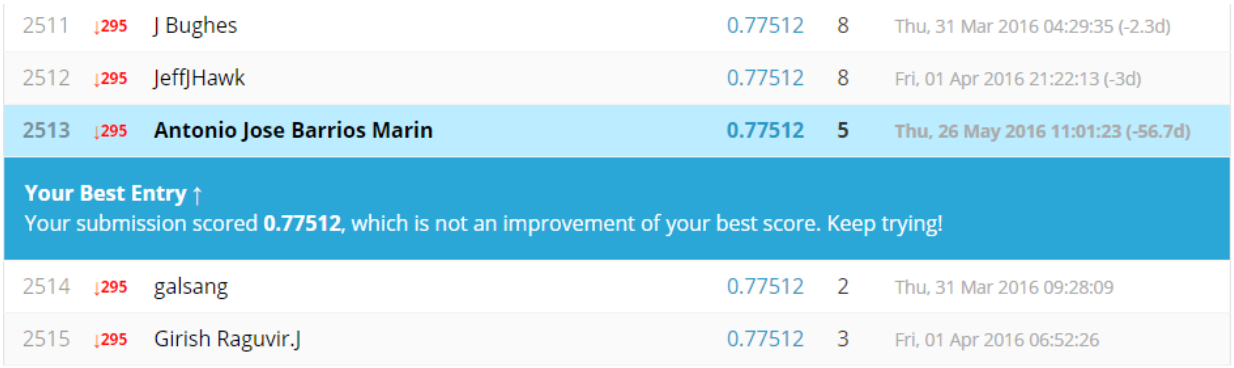

Figura 4: Posición alcanzada con el modelo LDA en la tabla de posiciones pública de la competición del Titanic. Esta figura muestra la posición alcanzada por un usuario en la competición, el nivel de precisión obtenido por el modelo, el número de resultados enviados a *Kaggle* y la fecha en la que se envió dicho resultado

Para intentar obtener un mejor modelo de la supervivencia al naufragio construimos un árbol de decisión a partir del conjunto de entrenamiento. Hay que tener en cuenta que, al ser numéricas, R trataría las variables `clase`, `sexo` y `supervivencia` como variables continuas, cuando en realidad son variables categóricas. Para hacer esto último explícito es por lo que se les aplica la función factor a cada una de ellas.

```
arbolent<-tree(factor(supervivencia) ~ edad + factor(clase) +
                 factor(sexo),  data = Titanicent)
```

El árbol de decisión obtenido es el siguiente:

```
arbolent
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 891 1187.00 0 ( 0.61616 0.38384 )
##    2) factor(sexo): female 314  358.50 1 ( 0.25796 0.74204 )
##      4) factor(clase): 3 144  199.60 1 ( 0.50000 0.50000 ) *
##      5) factor(clase): 1,2 170   70.41 1 ( 0.05294 0.94706 ) *
##    3) factor(sexo): male 577  559.30 0 ( 0.81109 0.18891 )
##      6) edad < 6.5 38   51.73 1 ( 0.42105 0.57895 ) *
##      7) edad > 6.5 539  476.50 0 ( 0.83859 0.16141 )
##       14) factor(clase): 2,3 422  286.50 0 ( 0.89336 0.10664 ) *
##       15) factor(clase): 1 117  152.80 0 ( 0.64103 0.35897 ) *
```

Una representación gráfica del árbol de decisión puede obtenerse de la siguiente manera:

```
plot(arbolent) # Dibuja la estructura del árbol
text(arbolent) # Etiqueta los nodos del árbol
title("Árbol de clasificación del conjunto Titanic")
```

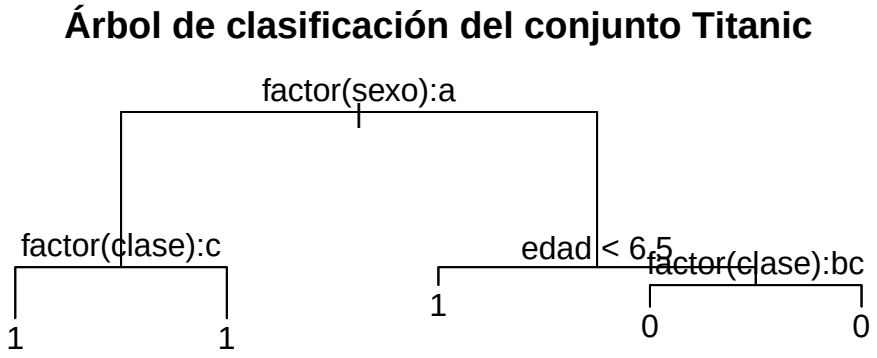

Figura 5: Representación de la supervivencia de los pasajeros del Titanic según las predicciones del árbol de clasificación

La figura 5 muestra un árbol de clasificación construido a partir del conjunto de datos de la competición Titanic, que contiene varias características de los pasajeros del Titanic. Según este árbol, un pasajero que sea mujer, independientemente de la clase a la que pertenezca es clasificado como `superviviente`. Por el contrario, solamente los varones menores de 6.5 años sobrevivirán según la clasificación de nuestro modelo.

El resultado que porporciona la función `predict` para un modelo de árbol de decisión y un conjunto de ejemplos es una matriz indicando, para cada ejemplo, la probabilidad de que su clasificación sea cada una de las clases que nos ocupa. A partir de esas probabilidades hay que elegir para cada ejemplo una clase, normalmente la más probable.

La competición del Titanic es una problema de clasificación binaria, por lo que para cada pasajero del conjunto de prueba debemos determinar si lo que el árbol de decisión predice más probable es la supervivencia o la no supervivencia. Usaremos la opción "class" que incorpora la función `predict` que extrae la clasificación para cada individuo del conjunto de datos.

```
prediccionarbol<-predict(arbolent, Titanitest, type = "class")
head(prediccionarbol, 10)
```

```
##  [1] 0 0 0 0 1 0 0 0 1 0
## Levels: 0 1
```

Para subir las predicciones obtenidas a *Kaggle* se guardan en un fichero con formato csv.

```
solucionarbol <- data.frame(PassengerID = titanictest$PassengerId,
                            Survived = prediccionarbol)
write.csv(solucionarbol, file = 'solucionarbol.csv', row.names = F)
```

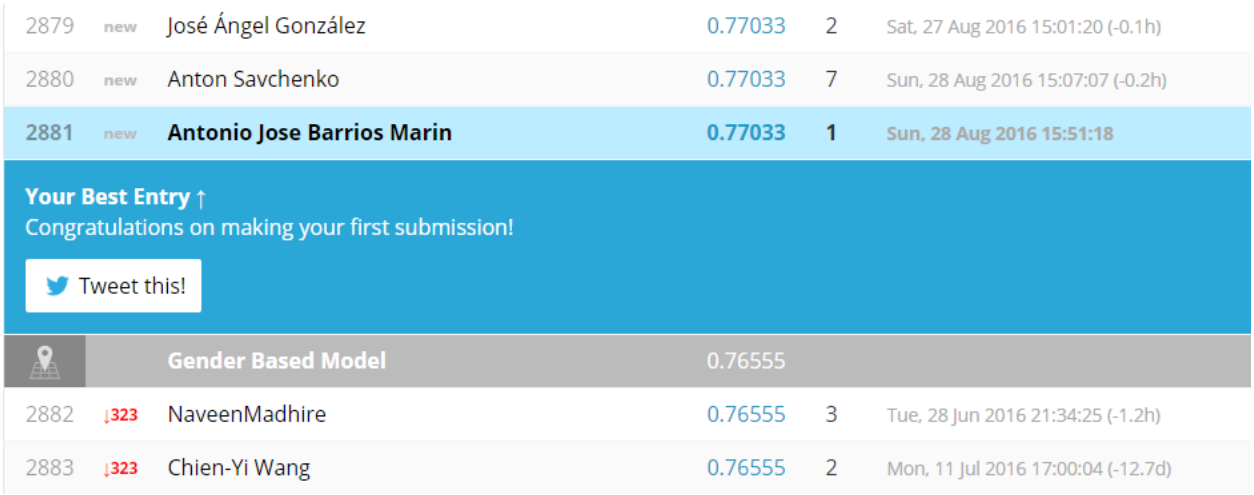

Figura 6: Posición alcanzada con el árbol de clasificación en la tabla de posiciones pública de la competición del Titanic

Obtenemos una puntuación de 0.77033, es decir, un 77.03 % de precisión lo que nos sitúa en la posición 2515 de la tabla de posiciones pública (figura 6).

Vemos que esta predicción obtenida no mejora la precisión obtenida previamente por el modelo LDA.

Por último, se ha construido un modelo *random forest*, tal y como recomienda la plataforma *Kaggle* para esta competición. Puesto que en esta metodología hay que realizar operaciones aleatorias, es necesario establecer una semilla aleatoria inicial para que los resultados sean reproducibles.

```
set.seed(12345) # Establecemos la semilla aleatoria
modelorf<- randomForest(factor(supervivencia) ~ edad + clase +
                          sexo, data = Titanicent)
modelorf
```

```
##
## Call:
##  randomForest(formula = factor(supervivencia) ~ edad + clase +      sexo, data = Tita
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 21.55%
## Confusion matrix:
##     0   1 class.error
## 0 488  61   0.1111111
## 1 131 211   0.3830409
```

La figura 7 es una representación gráfica del error cometido por nuestro modelo.

```
plot(modelorf)
legend('topright', colnames(modelorf$err.rate), col=1:3, fill=1:3)
```

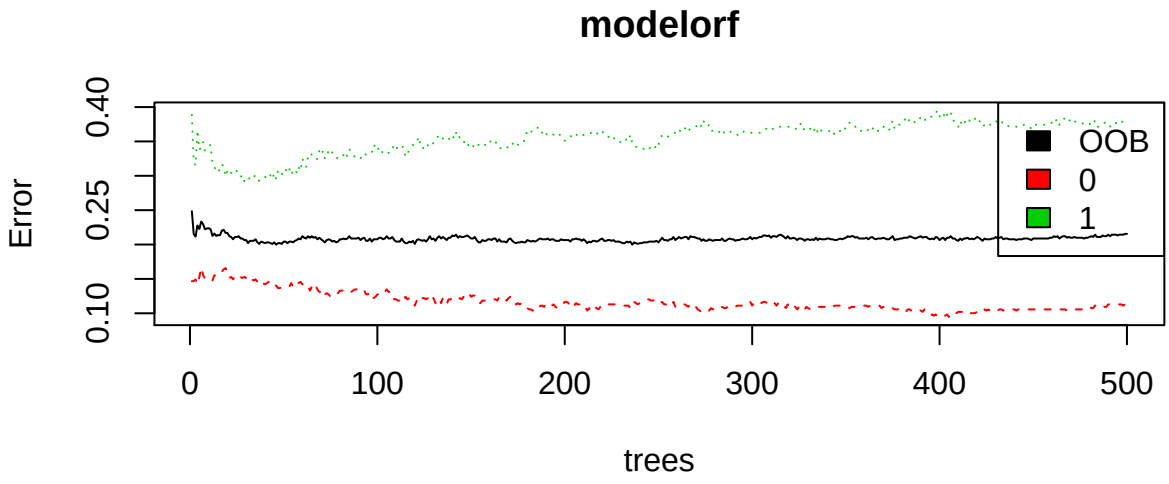

Figura 7: Representación del error producido en el random forest para cada grupo y el error OOB

Representamos la importancia de las variables del modelo, ordenadas de mayor a menor importancia mediante la figura 8.

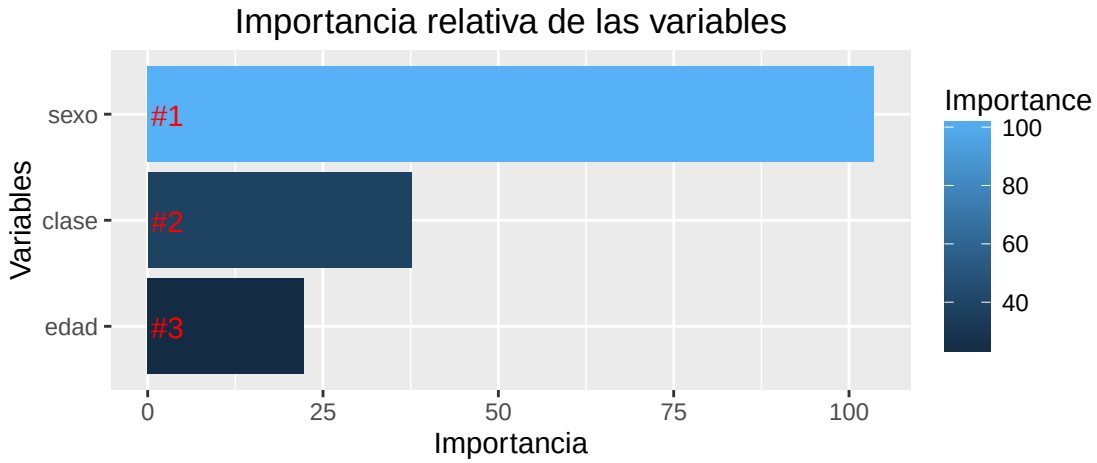

Figura 8: Representación de la importancia de las variables ordenadas de mayor a menor

Calculamos ahora la predicción que realiza el modelo sobre cada ejemplo del conjunto de prueba.

```
prediccionrf<-predict(modelorf,Titanitest)
head(prediccionrf,10)
```

```
##  1  2  3  4  5  6  7  8  9 10
##  0  0  0  0  1  0  1  0  0  0
## Levels: 0 1
```

En la figura 9 podemos ver una representación de la supervivencia predicha, por edad y sexo de los individuos.

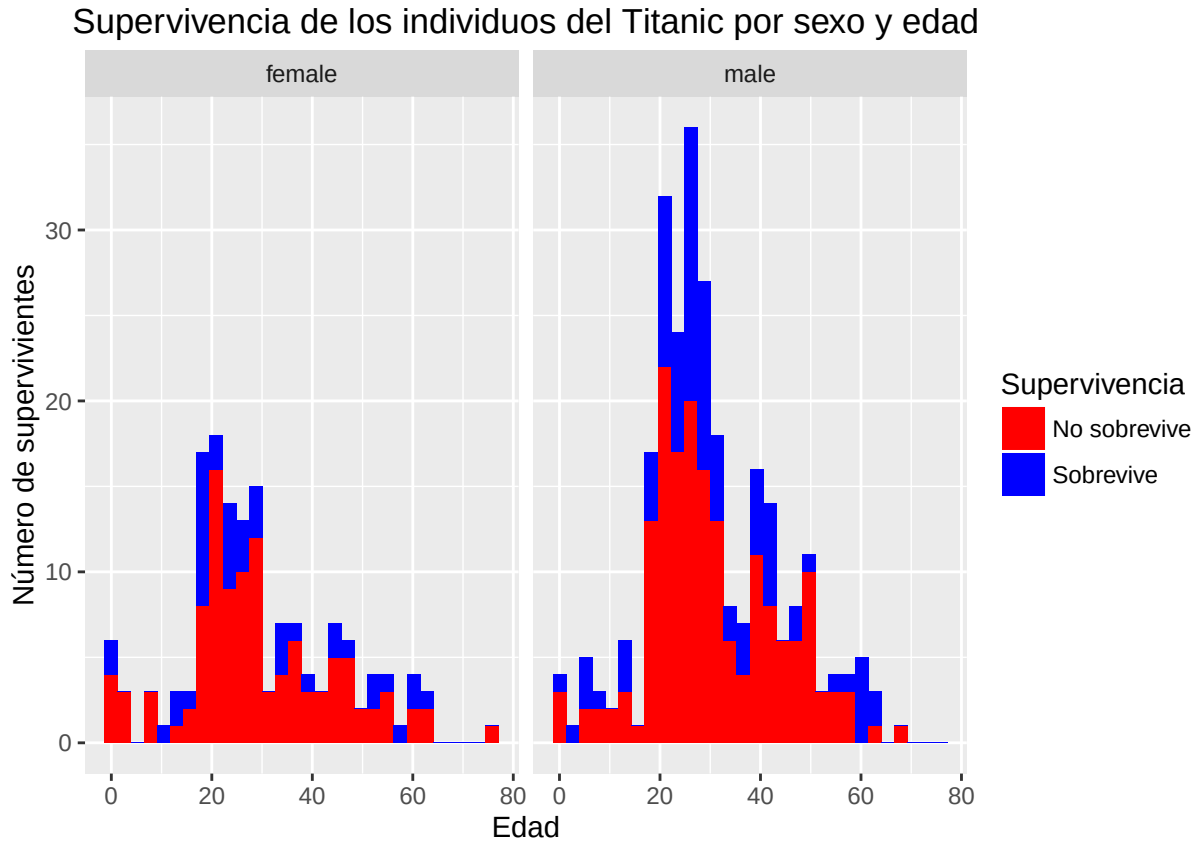

Figura 9: Representación de la supervivencia de los pasajeros del Titanic según las predicciones del modelo random forest

Para subir los resultados a *Kaggle* guardamos las predicciones obtenidas en un fichero con formato csv.

```
solucionrf <- data.frame(PassengerID = titanictest$PassengerId,
                         Survived = prediccionrf)
write.csv(solucionrf, file = 'solucionrf.csv', row.names = F)
```

Obtenemos una puntuación de 0.73684, es decir, un 73.68 % de precisión, lo que nos mantiene en la posición 2513 de la tabla de posiciones pública (figura 10).

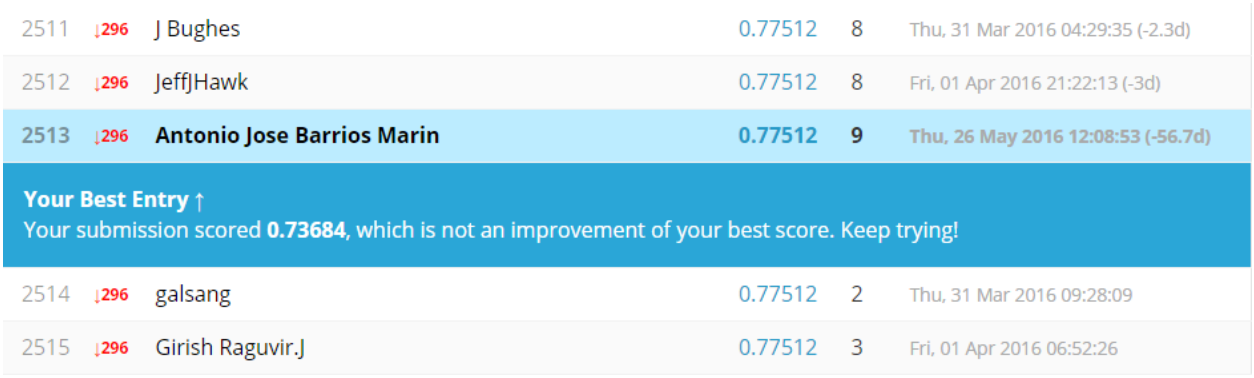

Figura 10: Posición alcanzada con el modelo *random forest* en la tabla de posiciones pública de la competición del Titanic

Una vía para mejorar nuestros modelos y, de esta forma, nuestra posición en la tabla de posiciones pública es considerar características adicionales a las ya tenidas en cuenta de edad, clase y sexo. Ampliaremos, por tanto, nuestro conjunto de características con las variables `Sibsp`, `parch` y `tarifa`. Hay que tener en cuenta que esta última tiene un valor perdido, que se ha imputado con el valor de la observación anterior.

```
lista <- list(supervivencia= titanicentrenamiento$Survived,
          edad= titanicentrenamiento$Age,
          clase= titanicentrenamiento$Pclass,
          sexo= titanicentrenamiento$Sex,
          sibsp=titanicentrenamiento$SibSp,
          parch=titanicentrenamiento$Parch,
          tarifa=titanicentrenamiento$Fare)
Titanicentcompleto<- as_data_frame(lista) %>% fill(edad) %>% fill(tarifa)
head(Titanicentcompleto,5)

Lista <- list(edad= titanictest$Age,
          clase= titanictest$Pclass,
          sexo= titanictest$Sex,
          sibsp=titanictest$SibSp,
          parch=titanictest$Parch,
          tarifa=titanictest$Fare)
Titanitestcompleto<- as_data_frame(Lista) %>% fill(edad) %>% fill(tarifa)
head(Titanitestcompleto,5)
```

Obtenemos entonces el siguiente análisis discriminante lineal.

```
ent.ldacompleto<- lda(supervivencia ~ edad + clase + sexo + sibsp +
                        parch + tarifa, data = Titanicentcompleto)
ent.ldacompleto
```

```
## Call:
## lda(supervivencia ~ edad + clase + sexo + sibsp + parch + tarifa,
##     data = Titanicentcompleto)
##
## Prior probabilities of groups:
##         0         1
## 0.6161616 0.3838384
##
## Group means:
##       edad    clase  sexomale     sibsp     parch   tarifa
## 0 30.33439 2.531876 0.8524590 0.5537341 0.3296903 22.11789
## 1 28.37307 1.950292 0.3187135 0.4736842 0.4649123 48.39541
##
## Coefficients of linear discriminants:
##                    LD1
## edad     -0.020555397
## clase    -0.688252725
## sexomale -2.176575053
## sibsp    -0.174133335
## parch    -0.083296056
## tarifa    0.001825976
```

Las predicciones del nuevo modelo sobre el conjunto de prueba son las siguientes:

```
prediccionldacompleta<-predict(ent.ldacompleto,
                               Titanitestcompleto)$class
head(prediccionldacompleta,10)
```

```
##  [1] 0 1 0 0 1 0 1 0 1 0
## Levels: 0 1
```

Vamos a representar esas predicciones en función del sexo y la edad de los pasajeros mediante la figura 11.

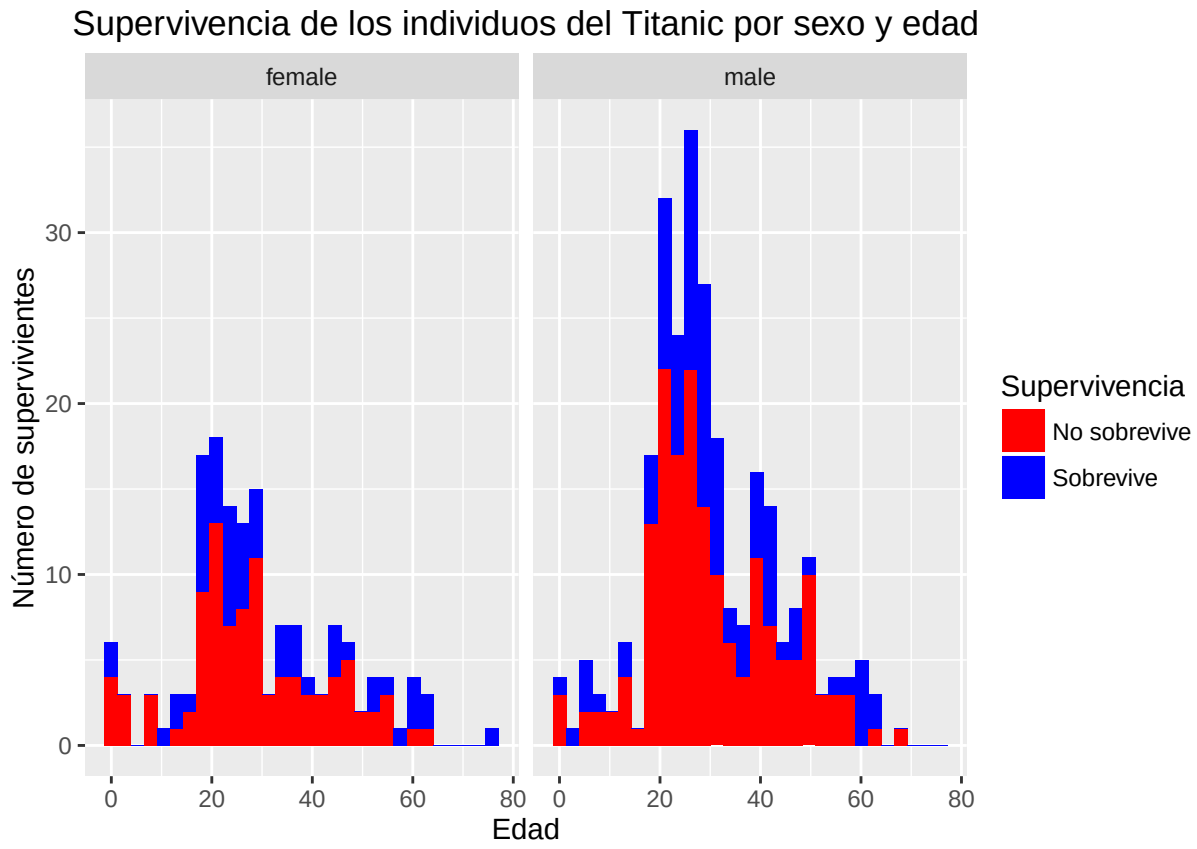

Figura 11: Representación de la supervivencia de los pasajeros del Titanic según las predicciones del modelo LDA

Para subir los resultados a *Kaggle* guardamos las predicciones obtenidas en un fichero con formato csv.

```
solucionldacompleta <- data.frame(PassengerID =titanictest$PassengerId,
                                  Survived = prediccionldacompleta )
write.csv(solucionldacompleta , file = 'solucionldacompleta .csv',
          row.names = F)
```

Finalmente, subimos los resultados a *Kaggle* y obtenemos una puntuación de 0.77990 (figura 12), es decir, un 77.99 % de precisión, lo que nos sitúa en la posición 2499 de la tabla de posiciones pública. Lo cual es una mejora en la precisión de 0.00478 con respecto a la obtenida previamente por el modelo LDA con menor número de variables.

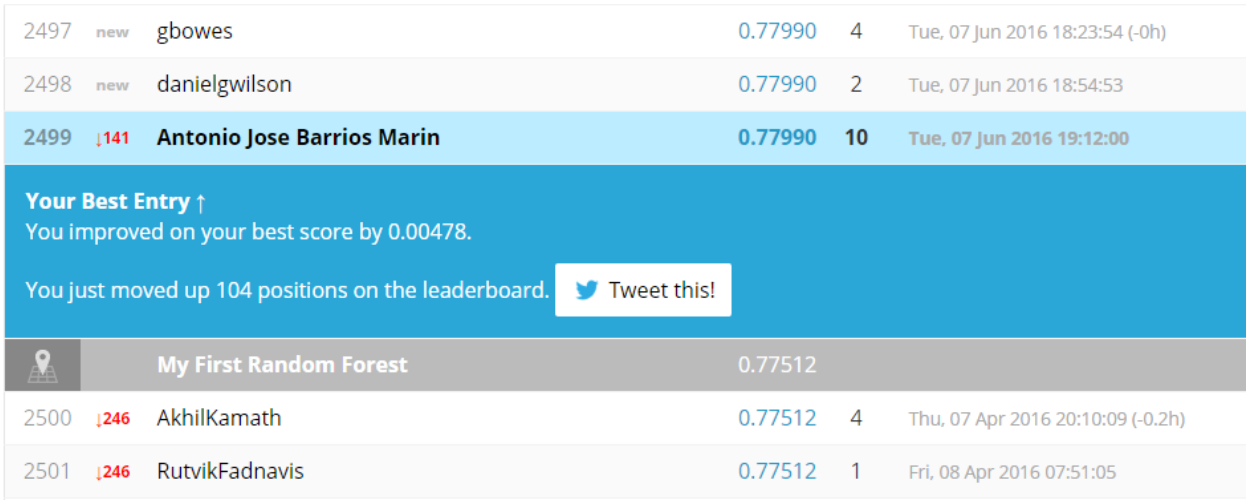

Figura 12: Posición alcanzada con el modelo LDA con el conjunto de características ampliado en la tabla de posiciones pública de la competición del Titanic

El nuevo árbol de decisión construido con el nuevo conjunto de características es el siguiente.

```
arbolentcompleto<-tree(factor(supervivencia) ~ edad + factor(clase)
                       + factor(sexo)  + factor(sibsp) + factor(parch)
                       + tarifa, data = Titanicentcompleto)
arbolentcompleto
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 891 1187.00 0 ( 0.61616 0.38384 )
##    2) factor(sexo): female 314  358.50 1 ( 0.25796 0.74204 )
##      4) factor(clase): 3 144  199.60 1 ( 0.50000 0.50000 )
##        8) tarifa < 23.35 117  158.40 1 ( 0.41026 0.58974 ) *
##        9) tarifa > 23.35 27   18.84 0 ( 0.88889 0.11111 ) *
##      5) factor(clase): 1,2 170   70.41 1 ( 0.05294 0.94706 ) *
##    3) factor(sexo): male 577  559.30 0 ( 0.81109 0.18891 )
##      6) edad < 6.5 38   51.73 1 ( 0.42105 0.57895 )
##       12) factor(sibsp): 0,3,4,5 27   36.50 0 ( 0.59259 0.40741 ) *
##       13) factor(sibsp): 1,2 11    0.00 1 ( 0.00000 1.00000 ) *
##      7) edad > 6.5 539  476.50 0 ( 0.83859 0.16141 )
##       14) factor(clase): 2,3 422  286.50 0 ( 0.89336 0.10664 ) *
##       15) factor(clase): 1 117  152.80 0 ( 0.64103 0.35897 ) *
```

En la representación gráfica de la figura 13 se observa claramente que las variables `tarifa` y `Sibsp` permiten una clasificación más precisa e influyen en nuestro modelo, mientras que la variable `parch` es irrelevante y no parece influir en el modelo.

```
plot(arbolentcompleto)
text(arbolentcompleto)
title("Árbol de clasificación con el conjunto ampliado")
```

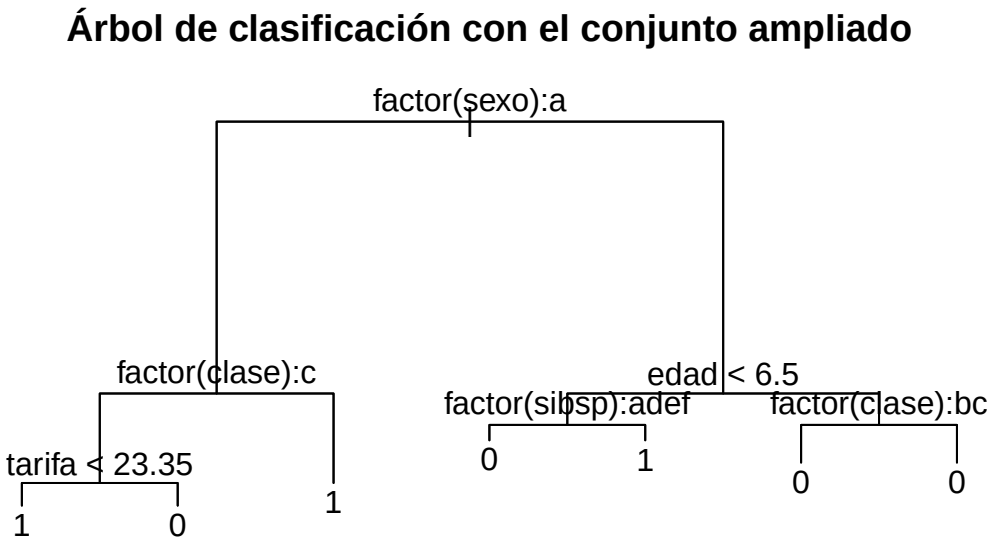

Figura 13: Representación de la supervivencia de los pasajeros del Titanic según un conjunto de características mayor

La figura muestra un árbol de clasificación construido a partir del conjunto de datos de la competición del Titanic, que contiene varias características de los pasajeros del Titanic. Según este árbol, un pasajero que sea mujer de tercera clase y que haya pagado menos de 23.35 por el tiquet es clasificada como `sobrevive`, mientras que si las mujeres de tercera clase han pagado más de 23.35 son clasificadas como `no sobrevive`. Las mujeres de 1ª y 2ª clase son clasificadas como `sobreviven`. En cuanto a los varones, todos aquellos mayores de 6.5 siguen siendo clasificados como `no sobreviven`. Por último, solo los varones menores de 6.5 años con 1 ó 2 hermanos/as, maridos o mujeres a bordo se clasifican como `sobreviven`.

Al igual que se hizo anteriormente, las predicciones que obtenemos a partir del árbol de decisión son la clasificación de los individuos en `sobrevive` o `no sobrevive`.

```
prediccionarbolcompleta<-predict(arbolentcompleto, Titanitestcompleto,
                                 type = "class")
head(prediccionarbolcompleta, 10)
```

```
##  [1] 0 1 0 0 1 0 1 0 1 0
## Levels: 0 1
```

Subimos, por tanto, los resultados a *Kaggle* obteniendo una puntuación de 0.7655, es decir, un 76.55 % de precisión, lo que nos sitúa en la posición 2498 de la tabla de posiciones pública (figura 14). Lo cuál no es una mejora con respecto a la mejora de precisión generada por el modelo LDA sobre el conjunto de nuevas características.

```
solucionarbolcompleta <-data.frame(PassengerID=titanictest$PassengerId,
                                   Survived = prediccionarbolcompleta )
write.csv(solucionarbolcompleta , file = 'solucionarbolcompleta .csv',
          row.names = F)
```

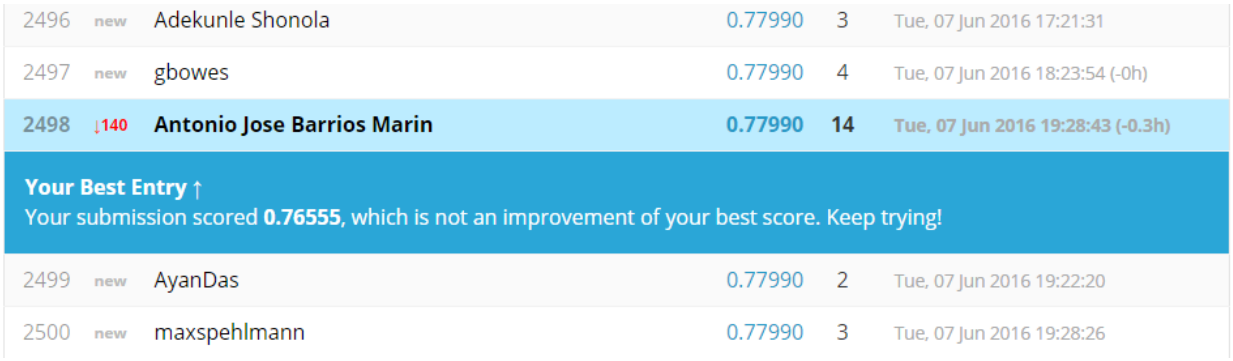

Figura 14: Posición alcanzada con el árbol de clasificación con el conjunto de características ampliado en la tabla de posiciones pública de la competición del Titanic

Por último, el modelo *random forest*, obtenido con el nuevo conjunto de características es el siguiente:

```
set.seed(12345)
modelorfcompleto<- randomForest(factor(supervivencia) ~ edad + clase
                                + sexo  + sibsp + parch + tarifa,
                                data = Titanicentcompleto)
```

```
modelorfcompleto
```

```
##
## Call:
##  randomForest(formula = factor(supervivencia) ~ edad + clase +      sexo + sibsp + pa
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 17.4%
## Confusion matrix:
##     0   1 class.error
## 0 499  50  0.09107468
## 1 105 237  0.30701754
```

En la representación gráfica del error cometido por el modelo (figura 15), se observa cómo ha disminuido tanto el error OOB como el error de clasificación en no superviviente y superviviente (dicha disminución es patente sobre todo en este último) con respecto al error cometido por el modelo *random forest* inicial.

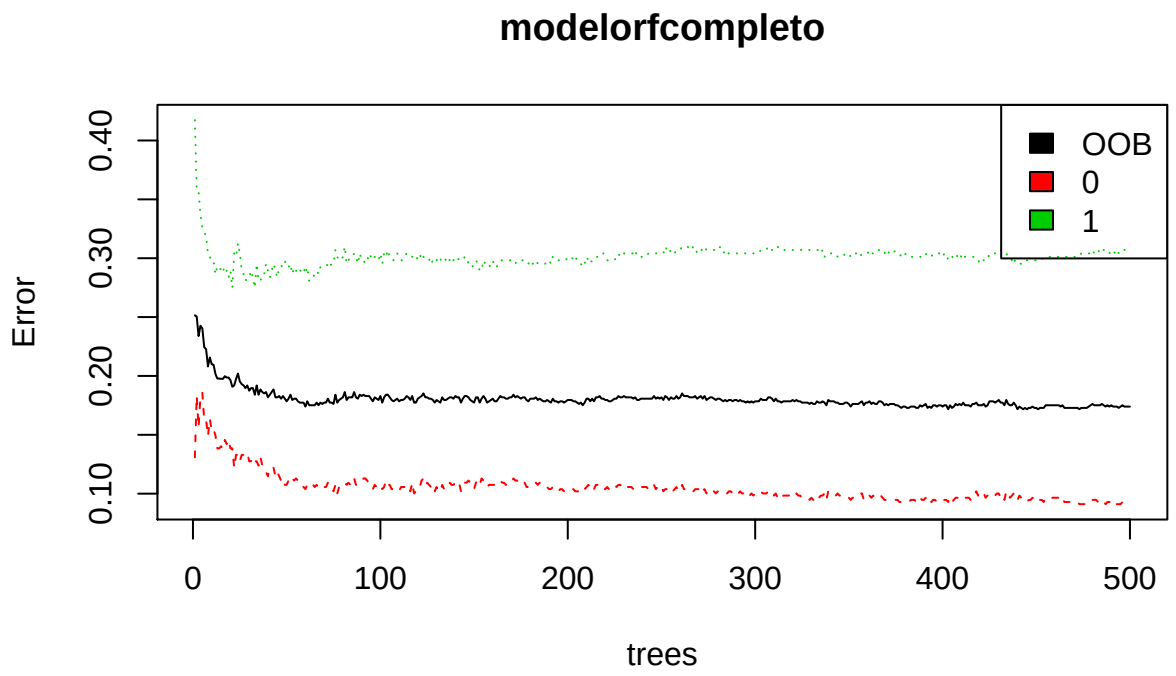

Figura 15: Representación del error cometido por cada grupo de la variable dependiente en el modelo random forest y del error OOB

La figura 16 representa el orden de importancia de las variables usadas en el modelo de mayor a menor importancia.

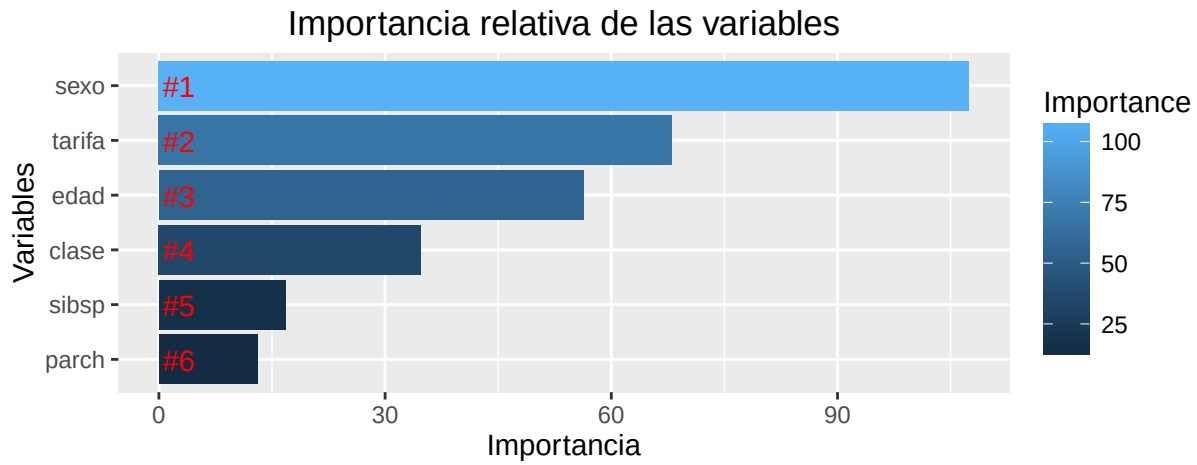

Figura 16: Representación de las variables predictoras del modelo random forest ordenadas de mayor a menor

Las predicciones que obtenemos sobre el conjunto de prueba son

```
prediccionrfcompleta<-predict(modelorfcompleto,Titanitestcompleto)
head(prediccionrfcompleta,10)
```

```
##  1  2  3  4  5  6  7  8  9 10
##  0  1  0  0  0  0  1  0  1  0
## Levels: 0 1
```

Finalmente subimos las predicciones calculadas previamente a *Kaggle* a través de un fichero .csv y obtenemos una puntuación de 0.77512, es decir, un 77.51 % de precisión, lo que nos sitúa en la posición 2498 de la tabla de posiciones pública (figura 17). Lo cual no es una mejora con respecto al modelo LDA sobre el conjunto de nuevas características.

```
solucionrfcompleta <- data.frame(PassengerID = titanictest$PassengerId,
                                 Survived = prediccionrfcompleta)
write.csv(solucionrfcompleta, file = 'solucionrfcompleta.csv',
          row.names = F)
```

Siendo la puntuación obtenida por el modelo *random forests* la que se muestra la figura 17.

2496	new	Adekunle Shonola	0.77990	3	Tue, 07 Jun 2016 17:21:31
2497	new	gbowes	0.77990	4	Tue, 07 Jun 2016 18:23:54 (-0h)
2498	**↓139**	**Antonio Jose Barrios Marin**	**0.77990**	**18**	**Tue, 07 Jun 2016 19:48:53 (-0.6h)**
Your Best Entry ↑ Your submission scored **0.77512**, which is not an improvement of your best score. Keep trying!					
2499	new	AyanDas	0.77990	2	Tue, 07 Jun 2016 19:22:20
2500	new	maxspehlmann	0.77990	3	Tue, 07 Jun 2016 19:28:26

Figura 17: Posición alcanzada con el modelo de *Random Forest* en el conjunto de características ampliado dentro de la tabla de posiciones pública de la competición del Titanic

Las predicciones enviadas a *Kaggle* previamente se representan por sexo y edad mediante la figura 18.

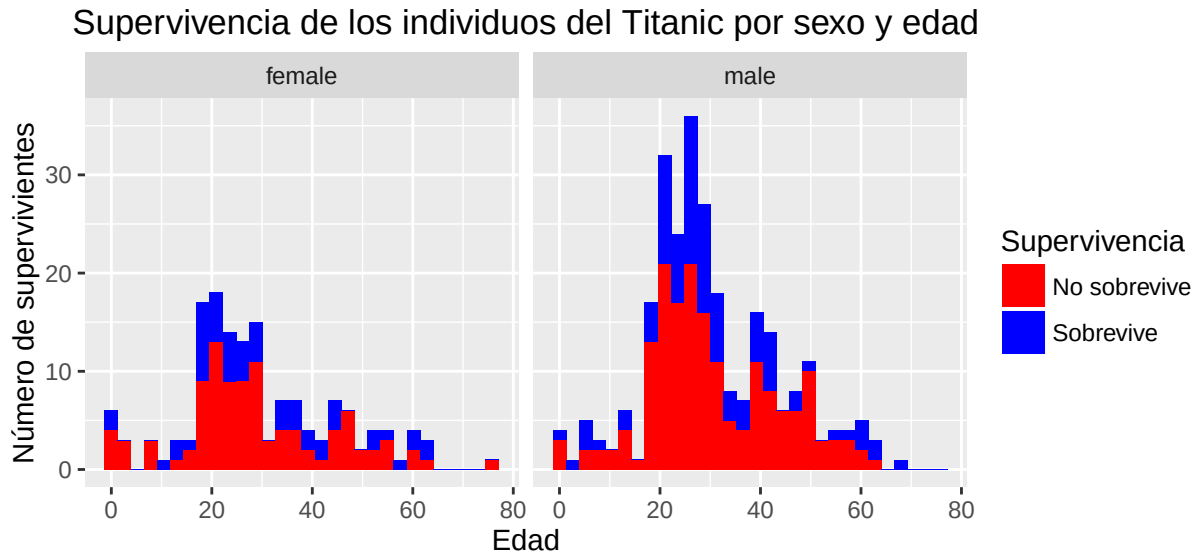

Figura 18: Representación de la supervivencia de los pasajeros del Titanic según un conjunto de características mayor en el modelo random forest

Competición animales de acogida (*shelter animals*)

Cada año, aproximadamente 7,6 millones de animales de compañía terminan en refugios estadounidenses. Muchos animales son abandonados por sus propietarios, mientras que otros son recogidos después de haberse perdido o sacados de situaciones de crueldad. Muchos de estos animales encuentran familias para siempre que se los llevan a casa, pero de igual manera muchos de ellos no son tan afortunados. 2,7 millones de perros y gatos son sacrificados en los EE.UU. cada año.

Usando un conjunto de datos con información sobre la admisión de mascotas en el Centro de Animales de Austin que incluye la raza, el color, el sexo y la edad, esta competición de *Kaggle* pide predecir el resultado de adopción para cada animal recogido.

Otro objetivo de la competición es ayudar a entender la tendencia en la acogida de animales. Esto podría ayudar a los refugios a centrar su energía en animales específicos que necesitan un poco más de ayuda para encontrar un nuevo hogar.

La competición de animales de acogida proporciona un conjunto de datos de entrenamiento con 26729 casos y otro de prueba con 11456 casos descritos por las siguientes 10 características:

- ID (Identificación del animal)
- Nombre
- Fecha de acogida
- Tipo de acogida
- Subtipo de acogida
- Tipo de animal
- Sexo
- Edad
- Raza
- Color

```
entanimal<-read.csv("trainanimal.csv",header=TRUE,sep=",")
head(entanimal,3)
```

```
##   AnimalID    Name            DateTime     OutcomeType OutcomeSubtype
## 1  A671945 Hambone 2014-02-12 18:22:00 Return_to_owner
## 2  A656520   Emily 2013-10-13 12:44:00      Euthanasia      Suffering
## 3  A686464  Pearce 2015-01-31 12:28:00        Adoption         Foster
##   AnimalType SexuponOutcome AgeuponOutcome                   Breed
## 1        Dog  Neutered Male         1 year  Shetland Sheepdog Mix
## 2        Cat  Spayed Female         1 year Domestic Shorthair Mix
## 3        Dog  Neutered Male        2 years           Pit Bull Mix
##         Color
## 1 Brown/White
## 2 Cream Tabby
## 3  Blue/White
```

```
testanimal<-read.csv("testanimal.csv",header=TRUE,sep=",")
head(testanimal,3)
```

```
##   ID     Name            DateTime AnimalType SexuponOutcome AgeuponOutcome
## 1  1   Summer 2015-10-12 12:15:00        Dog  Intact Female      10 months
## 2  2 Cheyenne 2014-07-26 17:59:00        Dog  Spayed Female        2 years
## 3  3      Gus 2016-01-13 12:20:00        Cat  Neutered Male         1 year
##                            Breed       Color
## 1         Labrador Retriever Mix   Red/White
## 2 German Shepherd/Siberian Husky   Black/Tan
## 3         Domestic Shorthair Mix Brown Tabby
```

Un primer análisis del conjunto de datos muestra que las variables predictoras presentan un gran número de modalidades. Esto, como veremos, resultará problemático a la hora de calcular los modelos de predicción.

Otra posible fuente de problema es el hecho de que el nombre de la variable identificadora de los animales es distinta en el conjunto de entrenamiento y en el de prueba. Además, en este último los identificadores son números, por lo que se ha decidido transformarlos a cadenas.

```
names(entanimal)[1] <- 'ID'
testanimal$ID<- as.character(testanimal$ID)
```

Vamos a unir los conjuntos de datos de entrenamiento y test para aplicar una serie de transformaciones a ambos conjuntos a la vez y así evitar que haya modalidades en un conjunto que no aparezcan en el otro y viceversa.

```
datosanimal<- bind_rows(entanimal,testanimal)
```

A continuación, nos hemos visto en la necesidad de simplificar las variables consideradas en nuestros modelos para que, o bien tengan menos modalidades, o bien se imputen los valores perdidos, o bien se puedan usar variables de difícil inclusión a priori.

La primera idea para construir un modelo predictivo es analizar si el nombre del animal influye en su situación final. Para ello, se ha usado el valor `Nameless` para los animales de los que no se conoce su nombre y se ha creado en el conjunto de datos otra variable `ConNombre`, con el valor 1 si se conocía el nombre y 0 si no.

```
datosanimal$Name <-ifelse(datosanimal$Name=="",'Nameless',
                          as.character(datosanimal$Name))
head(datosanimal$Name,10)
```

```
##  [1] "Hambone"  "Emily"    "Pearce"   "Nameless" "Nameless" "Elsa"
##  [7] "Jimmy"    "Nameless" "Lucy"     "Nameless"
```

```
datosanimal$ConNombre<-ifelse(datosanimal$Name!="Nameless",1,0)
head(datosanimal$ConNombre,10)
```

```
##  [1] 1 1 1 0 0 1 1 0 1 0
```

Ahora vamos a crear una nueva variable `sexo` para simplificar la variable `SexuponOutcome` con las modalidades: varón, hembra o desconocido.

```
datosanimal$Sex <- ifelse(grepl('Male', datosanimal$SexuponOutcome),
                          'Male',ifelse(grepl('Unknown',
                                              datosanimal$SexuponOutcome),
                                        'Unknown', 'Female'))
head(datosanimal$Sex,10)
```

```
##  [1] "Male"    "Female"  "Male"    "Male"    "Male"    "Female"  "Male"
##  [8] "Unknown" "Female"  "Female"
```

Puesto que el número de razas es excesivo, por un lado capturaremos si el animal es o no mestizo, creando para ello una nueva variable `Mestizo`.

```
datosanimal$Mestizo <- ifelse(grepl('Mix',datosanimal$Breed), 1, 0)
head(datosanimal$Mestizo,10)
```

```
##  [1] 1 1 1 1 0 0 1 1 1 0
```

Por otro lado, simplificaremos las razas mezcladas conservando únicamente la primera, consiguiendo de esa manera reducir el número de modalidades.

```
datosanimal$SimpleBreed <- sapply(datosanimal$Breed,
                            function(x) gsub(' Mix', '',
                            split = '/',fixed=FALSE)[[1]][1]))
head(datosanimal$SimpleBreed,10)
```

```
##  [1] "Shetland Sheepdog"         "Domestic Shorthair"
##  [3] "Pit Bull"                  "Domestic Shorthair"
##  [5] "Lhasa Apso"                "Cairn Terrier"
##  [7] "Domestic Shorthair"        "Domestic Shorthair"
##  [9] "American Pit Bull Terrier" "Cairn Terrier"
```

De la misma forma que hemos hecho con la raza, vamos a hacer ahora para los colores de los animales, donde nos quedaremos tan sólo con el primer color de cada animal.

```
datosanimal$SimpleColor <- sapply(datosanimal$Color,
                                  function(x)strsplit(as.character(x),
                                     split = '/| ')[[1]][1])
head(datosanimal$SimpleColor,10)
```

```
##  [1] "Brown" "Cream" "Blue"  "Blue"  "Tan"   "Black" "Blue"  "Brown"
##  [9] "Red"   "White"
```

Otra variable que vamos a usar es `DateTime`, que transformaremos mediante el paquete `lubridate`, separando la hora de la fecha. Posteriormente, a partir de la hora del día crearemos una variable con las partes del día (mañana, mediodía, tarde y noche).

```
datosanimal$Hour <- hour(datosanimal$DateTime)

datosanimal$TimeofDay <- ifelse(datosanimal$Hour > 5 &
              datosanimal$Hour < 11, 'morning',
                ifelse(datosanimal$Hour > 10 & datosanimal$Hour < 16,
                 'midday',ifelse(datosanimal$Hour > 15 &
                    datosanimal$Hour < 20, 'lateday','night')))
datosanimal$TimeofDay <- factor(datosanimal$TimeofDay,
                  levels = c('morning', 'midday','lateday', 'night'))
head(datosanimal$TimeofDay,10)
```

```
##  [1] lateday midday  midday  lateday midday  midday  midday  lateday
##  [9] lateday morning
## Levels: morning midday lateday night
```

La última variable que consideramos y que debemos transformar es la edad de los animales. Para ello, a partir de la edad en el momento de su adopción (que está formada por: un número y una cadena que indica un período de tiempo) calculamos esa misma edad como número de días, para que todos esos períodos de tiempo estén en la misma escala.

```
datosanimal$TimeValue <- sapply(datosanimal$AgeuponOutcome,
                   function(x) strsplit(as.character(x),' ')[[1]][1])

datosanimal$UnitofTime <- sapply(datosanimal$AgeuponOutcome,
                   function(x) strsplit(as.character(x),' ')[[1]][2])
datosanimal$UnitofTime <- gsub('s', '', datosanimal$UnitofTime)
datosanimal$TimeValue  <- as.numeric(datosanimal$TimeValue)
datosanimal$UnitofTime <- as.factor(datosanimal$UnitofTime)
```

```
numerodias <- ifelse(datosanimal$UnitofTime == 'day', 1,
               ifelse(datosanimal$UnitofTime == 'week', 7,
                ifelse(datosanimal$UnitofTime == 'month', 30,
                  ifelse(datosanimal$UnitofTime == 'year', 365, NA))))
datosanimal$AgeinDays <- datosanimal$TimeValue * numerodias
```

Una vez que se ha calculado la edad de las mascotas en días, hay varios valores perdidos y para evitarlos vamos a imputarlos por el valor del caso anterior.

```
sum(is.na(datosanimal$AgeinDays))
```

```
## [1] 24
```

```
datosanimal <- datosanimal %>% fill(AgeinDays)

sum(is.na(datosanimal$AgeinDays))
```

```
## [1] 0
```

Finalmente, convertimos a factores todas las variables consideradas, para que los algoritmos de construcción de modelos predictivos las traten como variables categóricas.

```
factorVars <- c('Name','OutcomeType','OutcomeSubtype','AnimalType',
                'SexuponOutcome','AgeuponOutcome','SimpleBreed',
                'SimpleColor','ConNombre','Mestizo','Sex','TimeofDay')

datosanimal[,factorVars] <- lapply(datosanimal[,factorVars],
                                   function(x) as.factor(x))
```

Una vez que se han tratado todas las variables que se pretenden utilizar en el modelo para predecir el tipo de acogida de los animales, separamos los datos en datos de entrenamiento y de prueba.

```
entanimal<- datosanimal[1:26729,]
testanimal<- datosanimal[26730:nrow(datosanimal),]
```

Estamos ya en condiciones de entrenar modelos que, a partir de las variables `AnimalType`, `Sex`, `AgeinDays`, `ConNombre`, `Hour`, `TimeofDay`, `SimpleColor`, `SimpleBreed` y `Mestizo` predigan el valor de la variable `OutcomeType`.

En primer lugar, se ha aplicado un análisis discriminante lineal, una técnica de clasificación que en principio tiene poco sentido porque como ya se explicó anteriormente este modelo de clasificación es adecuado para variables continuas. Sin embargo, en este marco de datos tenemos variables categóricas como el `Sexo` o `SimpleColor` entre otras.

A pesar de este inconveniente se ha decidido continuar con el modelo sin esperar buenos resultados a priori.

```
animallda<- lda(OutcomeType ~ AnimalType + Sex + AgeinDays
                + ConNombre + Hour + TimeofDay + SimpleColor
                + Mestizo,data = entanimal)
```

Vamos a calcular las predicciones del modelo sobre cada ejemplo del conjunto de prueba. Para esta competición nos interesa las probabilidades de pertenencia a cada clase.

```
prediccionldaanimal<-predict(animallda,testanimal)$posterior
head(prediccionldaanimal,5)
```

```
##        Adoption         Died  Euthanasia Return_to_owner   Transfer
## 26730 0.4567781 0.0019228809 0.016688682      0.20295124 0.32165913
## 26731 0.7157488 0.0005622990 0.009616648      0.18899448 0.08507778
## 26732 0.4378212 0.0062182508 0.026554651      0.07933282 0.45007312
## 26733 0.7350660 0.0003587081 0.005916812      0.17459189 0.08406660
## 26734 0.5947599 0.0006752925 0.016945679      0.29598616 0.09163299
```

Para subir los resultados a *Kaggle* guardamos las predicciones obtenidas en un fichero con formato csv. Este fichero está compuesto por varias columnas: la primera es el ID del animal y el resto de columnas representa la probabilidad de que cada animal reciba un tipo u otro de adopción.

```
solucionldaanimal <- data.frame(ID = testanimal$ID,
                                Adoption= prediccionldaanimal[,1],
                                Died= prediccionldaanimal[,2],
                                Euthanasia= prediccionldaanimal[,3],
                                Return_to_owner= prediccionldaanimal[,4],
                                Transfer = prediccionldaanimal[,5])
write.csv(solucionldaanimal, file = 'solucionldaanimal.csv',
          row.names = F)
```

Obtenemos una puntuación de 1.00582, lo que nos sitúa en la posición 835 de la tabla de posiciones pública (figura 19). Para conocer la posición final en la tabla de posiciones privada se debería esperar a que la competición termine. En este caso, el plazo termina el 31 de julio de 2016. Cabe destacar que en esta competición, cuanto menor sea la puntuación obtenida mejor

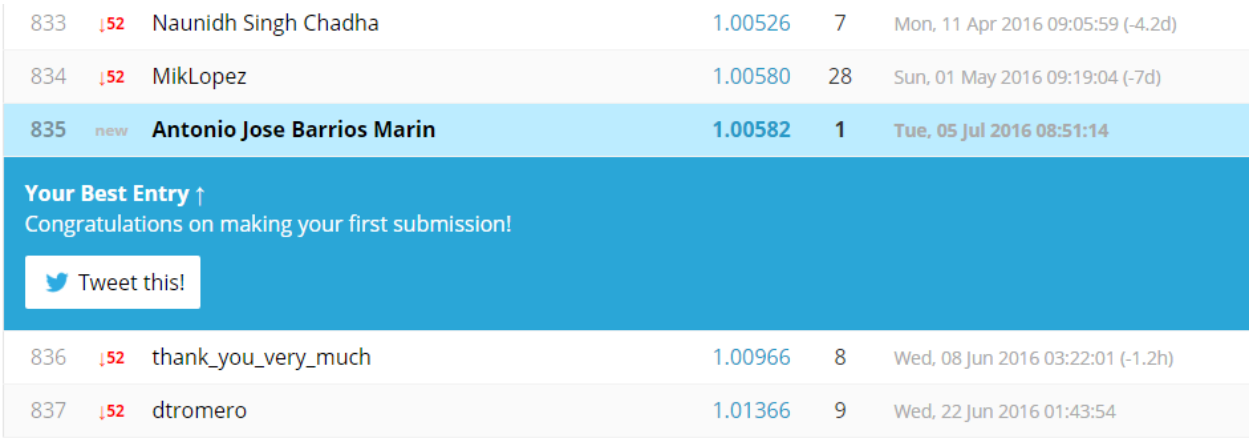

Figura 19: Posición alcanzada con el modelo LDA en el conjunto de características ampliado dentro de la tabla de posiciones pública de la competición *shelter animals*

Para intentar obtener una mejor clasificación de la adopción de los animales se va a construir un árbol de decisión.

```
arbolanimal<-tree(OutcomeType ~ AnimalType + Sex + AgeinDays
                  + ConNombre + Hour + TimeofDay + SimpleColor
                  + Mestizo, data = entanimal)
arbolanimal
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 26729 66470 Adoption ( 0.402896 0.007370 0.058177 0.179056 0.352501 )
##    2) ConNombre: 0 7691 15360 Transfer ( 0.218177 0.015603 0.105968 0.019893 0.640359
##      4) AgeinDays < 29 2102  1345 Transfer ( 0.000000 0.020457 0.034729 0.013321 0.93
##      5) AgeinDays > 29 5589 12400 Transfer ( 0.300233 0.013777 0.132761 0.022365 0.53
##       10) Hour < 16.5 3820  7771 Transfer ( 0.162827 0.016754 0.153141 0.021728 0.645
##       11) Hour > 16.5 1769  3556 Adoption ( 0.596947 0.007349 0.088751 0.023742 0.283
##    3) ConNombre: 1 19038 45170 Adoption ( 0.477519 0.004045 0.038870 0.243355 0.23621
##      6) AgeinDays < 255 6111 10440 Adoption ( 0.677467 0.008182 0.011946 0.050401 0.2
##       12) Hour < 16.5 3586  6740 Adoption ( 0.552984 0.012549 0.014222 0.040156 0.380
##       13) Hour > 16.5 2525  2788 Adoption ( 0.854257 0.001980 0.008713 0.064950 0.070
##      7) AgeinDays > 255 12927 31990 Adoption ( 0.382997 0.002089 0.051597 0.334571 0.
##       14) Hour < 16.5 8760 22260 Adoption ( 0.330936 0.002397 0.058676 0.312215 0.295
##       15) Hour > 16.5 4167  8841 Adoption ( 0.492441 0.001440 0.036717 0.381569 0.087
```

Una representación gráfica de nuestro árbol de decisión puede obtenerse de la siguiente manera:

```
plot(arbolanimal) #Dibuja la estructura del árbol
text(arbolanimal) #Etiqueta los nodos del árbol
title("Árbol de clasificación de la acogida de los animales")
```

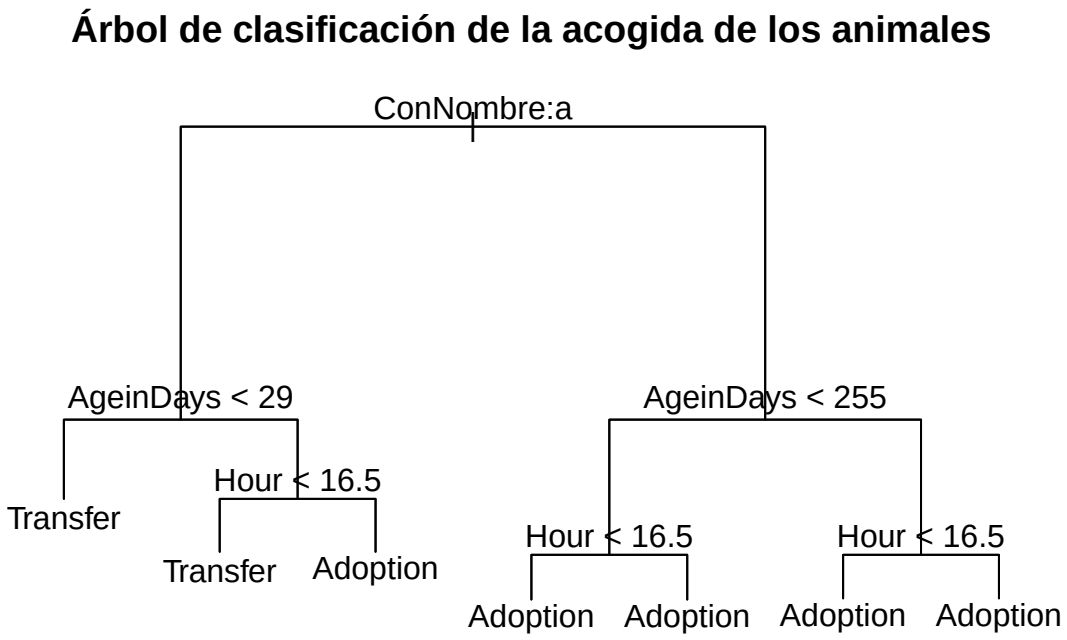

Figura 20: Representación de la acogida de los animales del centro de animales de Austin según un conjunto de características

La figura 20 muestra un árbol de clasificación construido a partir del conjunto de datos de la competición *shelter animals*, que contiene varias características de los animales de un centro. Según este árbol, un animal sin nombre y menor de 29 días es clasificado como `transferido`, si además, es antes de las 16:30 es clasificado también como `transferido`. Por el contrario, si es más tarde de esa hora es clasificado como `adoptado`. Sin embargo, todos los animales con nombre son clasificados como `adoptados`.

Obtenemos las siguientes probabilidades de pertenecer a cada uno de los grupos de nuestra variable dependiente de dicho modelo sobre cada ejemplo del conjunto de prueba.

```
predicionarbolanimal<-predict(arbolanimal,testanimal)
head(predicionarbolanimal,5)
```

```
##        Adoption        Died  Euthanasia Return_to_owner   Transfer
## 26730 0.3309361 0.002397260 0.058675799       0.3122146 0.29577626
## 26731 0.4924406 0.001439885 0.036717063       0.3815695 0.08783297
## 26732 0.3309361 0.002397260 0.058675799       0.3122146 0.29577626
## 26733 0.8542574 0.001980198 0.008712871       0.0649505 0.07009901
## 26734 0.4924406 0.001439885 0.036717063       0.3815695 0.08783297
```

Guardamos las probabilidades obtenidas en un fichero csv. para subir los resultados a *Kaggle*.

```
solucionarbolanimal <- data.frame(ID = testanimal$ID,
                                  Adoption = predicionarbolanimal[,1],
                                  Died = predicionarbolanimal[,2],
                                  Euthanasia = predicionarbolanimal[,3],
                                  Return_to_owner = predicionarbolanimal[,4],
                                  Transfer = predicionarbolanimal[,5])
write.csv(solucionarbolanimal, file = 'solucionarbolanimal.csv',
          row.names = F)
```

Para el árbol de decisión obtenemos una puntuación de 1.00143, lo que significa una mejora de 0.00439 respecto al resultado obtenido previamente por el LDA (figura 21). Lo que nos sitúa en la posición 847 de la tabla de posiciones pública.

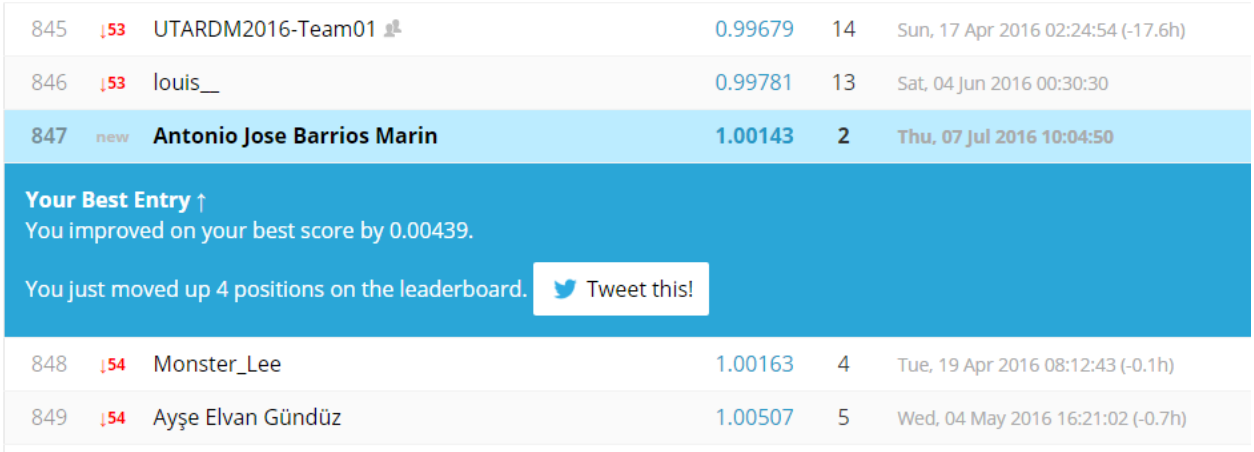

Figura 21: Posición alcanzada con el árbol de clasificación en el conjunto de características ampliado dentro de la tabla de posiciones pública de la competición *shelter animals*

Por último, se aplicará un modelo *random forest*, ya que es un método generalmente más preciso y potente que el árbol de decisión. Para solucionar el hecho de que el método *random forest* realiza operaciones aleatorias, es necesario establecer una semilla aleatoria antes de elaborar el modelo para que los resultados sean reproducibles.

```
set.seed(12345)
modelorfanimal<- randomForest(OutcomeType ~ AnimalType + Sex
                              + AgeinDays + ConNombre + Hour
                              + TimeofDay + SimpleColor + Mestizo,
                              data = entanimal)
modelorfanimal
```

```
##
## Call:
##  randomForest(formula = OutcomeType ~ AnimalType + Sex + AgeinDays +      ConNombre +
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 37.41%
## Confusion matrix:
##                 Adoption Died Euthanasia Return_to_owner Transfer
## Adoption            8592    0         16            1101     1060
## Died                  56    0          8               8      125
## Euthanasia           470    0        138             287      660
## Return_to_owner     2365    0          9            2068      344
## Transfer            2607    2         33             847     5933
##                 class.error
## Adoption          0.2021543
## Died              1.0000000
## Euthanasia        0.9112540
## Return_to_owner   0.5679064
## Transfer          0.3703035
```

La figura 22 es una representación gráfica del error OOB.

```
plot(modelorfanimal)
legend('topright', colnames(modelorfanimal$err.rate), col=1:6,
       fill=1:6)
```

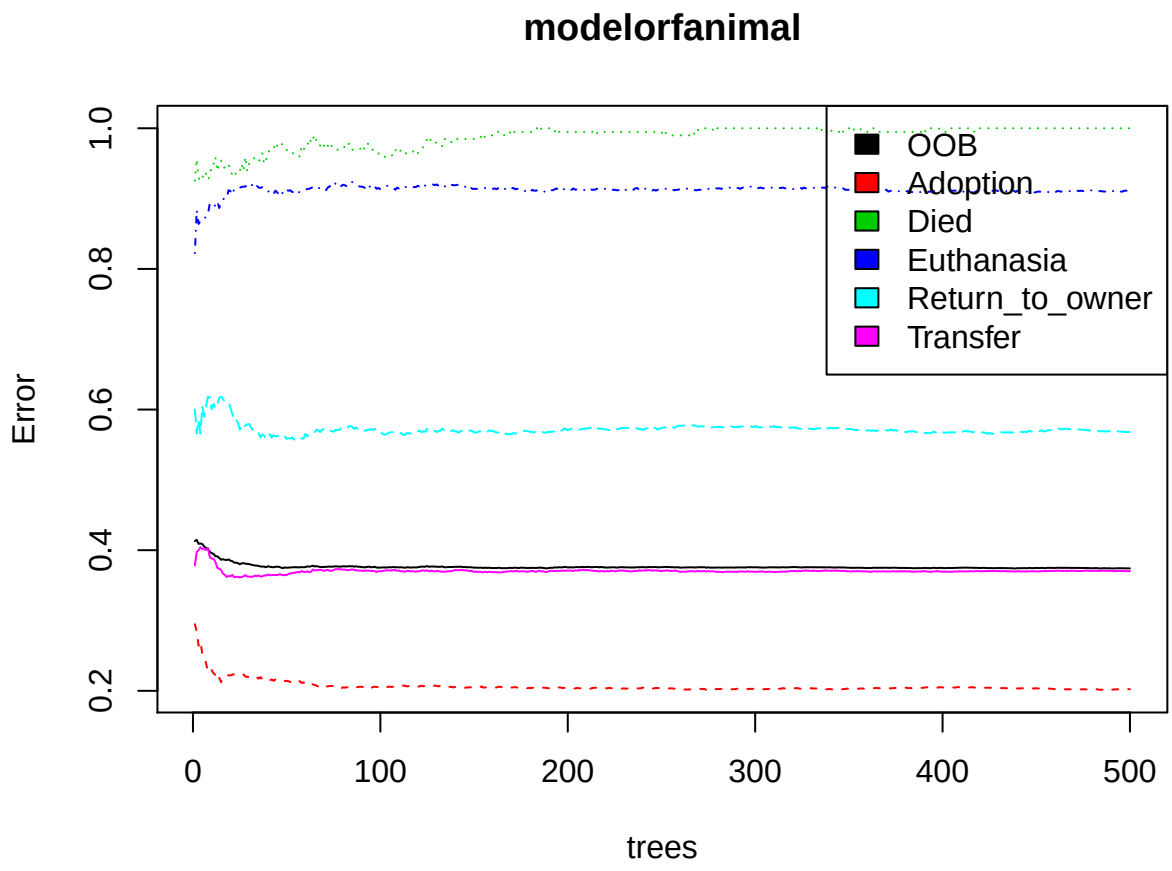

Figura 22: Representación del error producido en cada grupo de la variable dependiente y del error OOB

Una representación de la importancia de las variables del modelo, ordenadas de mayor a menor importancia se muestra en la figura 23.

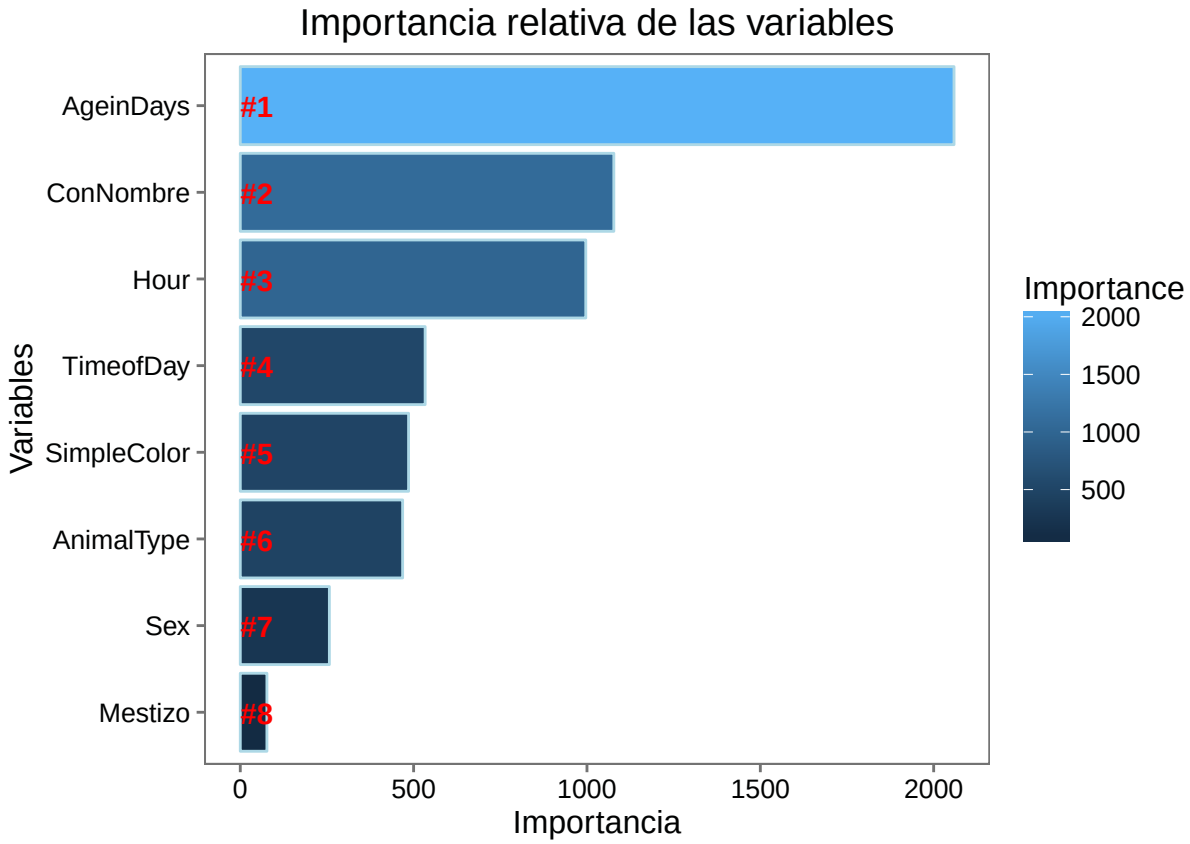

Figura 23: Representación de la importancia de las variables usadas en el modelo random forest mostradas en orden descendente

Una vez que tenemos un ranking de importancia de las variables utilizadas, se van a calcular las probabilidades que realiza el modelo sobre cada ejemplo del conjunto de prueba.

```
predicionrfanimal<-predict(modelorfanimal, testanimal, type = "prob")
head(predicionrfanimal, 5)
```

```
##       Adoption Died Euthanasia Return_to_owner Transfer
## 26730    0.638    0          0           0.086    0.276
## 26731    0.984    0          0           0.016    0.000
## 26732    0.312    0          0           0.014    0.674
## 26733    0.926    0          0           0.070    0.004
## 26734    0.618    0          0           0.382    0.000
```

Para subir los resultados a *Kaggle* se guardan dichas predicciones en un fichero csv.

```
solucionrfanimal <- data.frame(ID = testanimal$ID,
                               Adoption = predicionrfanimal[,1],
                               Died = predicionrfanimal[,2],
                               Euthanasia = predicionrfanimal[,3],
                               Return_to_owner = predicionrfanimal[,4],
                               Transfer = predicionrfanimal[,5])
write.csv(solucionrfanimal, file = 'solucionrfanimal.csv', row.names = F)
```

Para el modelo *random forest* obtenemos una puntuación de 2.00670, lo que nos mantiene en la posición 847 de la tabla de posiciones pública alcanzada por el modelo anterior (figura 24). Este resultado obtenido es mucho mayor que el obtenido por cualquiera de los otros dos métodos, lo cual es negativo puesto que se trata de minimizar el error. Este objetivo se alcanza con el árbol de decisión que se construyó previamente.

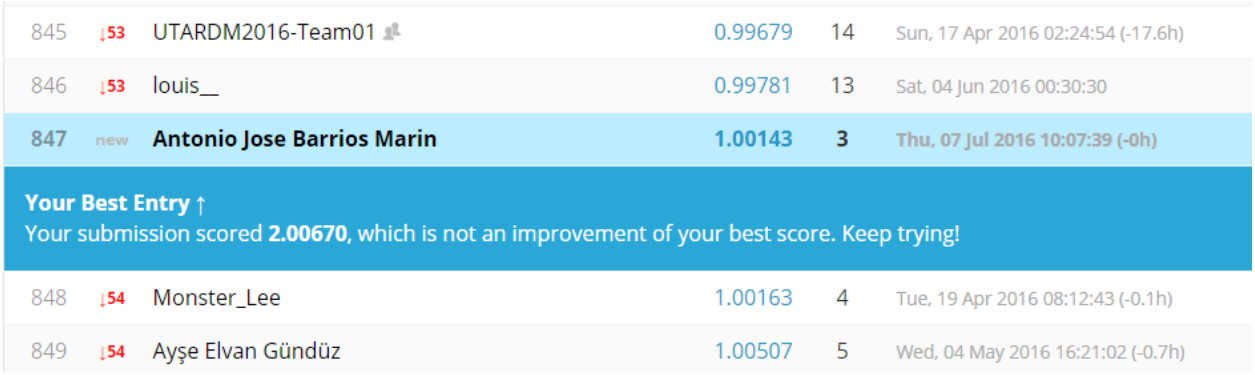

Figura 24: Posición alcanzada con el modelo de *Random Forest* en el conjunto de características dentro de la tabla de posiciones pública de la competición *shelter animals*

Para finalizar el estudio de esta competición vamos a realizar un análisis gráfico en el que observando el ranking de importancia de las variables, que devuelve el modelo del *random forests*, comparando una variable que tiene gran importancia (`AgeinDays`), una que tiene relativamente poca importancia (`TimeofDay`) y otra que tiene muy poca importancia (`Mestizo`) con la clasificación que se produce en la recogida de los animales en el modelo que mejores resultados proporciona que es el árbol de decisión.

Capítulo 5: Herramientas utilizadas

En este capítulo se van a describir brevemente las herramientas que se han utilizado para la realización de este proyecto.

Como herramienta principal para el tratamiento de los datos de las competiciones y para escribir las memorias del proyecto se ha utilizado el software estadístico *R*, ya que incluye una gran cantidad de paquetes para aplicar algoritmos de aprendizaje automático. Se ha usado el editor RStudio que da mayor comodidad al usuario separando la pantalla en cuatro apartados: uno para escribir el código fuente, otro para la consola y ejecutar el código, otro que guarda todos los objetos creados en el espacio de trabajo y la última partición que incluyen archivos a cargar del directorio en el que se trabaja, gráficos ejecutados, paquetes disponibles y para instalar y la ayuda de *R*.

Paquetes de *R* utilizados

El resto de herramientas que se utilizaron en el proyecto fueron "paquetes" de *R* para ejecutar modelos que no implementa de serie *R*, para procesar datos de las competiciones o para escribir archivos dinámicos.

R Markdown

Este paquete permite escribir documentos dinámicos y presentaciones a través de *R*. Esta herramienta combina sintaxis de markdown (una manera sencilla de escribir en formato texto) con código de *R* que puede incluir en el documento final el código ejecutado y su resultado.

Paquete *MASS* (https://cran.r-project.org/web/packages/MASS)

Este paquete provee un gran número de funciones y conjuntos de datos basados en el libro *"Modern Applied Statistics with S, MASS"* escrito por W.N. Venables y B.D. Ripley.

Paquete *tree* (http://cran.r-project-org/web/packages/tree)

Este paquete permite construir y manejar árboles de clasificación y regresión (*CART*).

Paquete *randomForest* (https://cran.r-project.org/web/packages/randomForest/)

Este paquete implementa el modelo de selvas aleatorias (*random forest*) para problemas de clasificación y regresión.

Paquete dplyr (https://cran.r-project.org/web/packages/dplyr/)

Este paquete proporciona una herramienta consistente y eficiente para trabajar con marcos de datos.

Paquete tidyr (https://cran.r-project.org/web/packages/tidyr/)

Paquete diseñado específicamente para la estructuración ordenada de conjuntos de datos. Se integra bien con los procesos realizados por el paquete `dplyr` mencionado anteriormente.

Paquete magrittr (https://cran.r-project.org/web/packages/magrittr/)

Este paquete provee un mecanismo para unir comandos con un nuevo operador, %>%. Este operador reenvía el resultado de una expresión a la siguiente expresión o llamada de función. Esto sirve de apoyo para la expresión situada a la derecha simplificando el código a través de este operador.

Paquete lintr (http://cran.r-project-org/web/packages/lintr)

Este paquete da recomendaciones sobre estilo, errores de sintaxis y posibles errores semánticos en un documento. Se integra perfectamente con el editor *RStudio*.

Paquete ggplot2 (http://ggplot2.org https://cran.r-project.org/web/packages/ggplot2)

GGplot2 es un sistema gráfico para *R*, basado en la grmática de gráficos desarrollada por L. Wilkinson. Proporciona un potente modelo que facilita la creación de complejos gráficos multicapas, a la vez que se ocupa de muchos de los detalles que hacen de la construcción de gráficos una molestia.

Paquete lubridate (https://cran.r-project.org/web/packages/lubridate/)

Este paquete implementa funciones para manipular fechas y horas de manera sencilla.

Conclusiones

El principal objetivo de este TFG ha sido el de introducirse al mundo del aprendizaje automático o *machine learning*, un campo en el que aparecen constantemente nuevos métodos que permiten resolver con más eficiencia los problemas que se pueden plantear en el día a día de una empresa, institución pública, etc. El otro gran objetivo del trabajo, era aplicar técnicas ya conocidas para buscar solución a algunos de los problemas presentados en la plataforma web *Kaggle y tratar de quedar en buena posición al final de las competiciones en las que se participó.

Teniendo en mente estos objetivos, a través del software *R* se ha aprendido a aplicar algunos algoritmos para satisfacer las necesidades de las competiciones planteadas, básicamente problemas de clasificación. Además, se ha encontrado con algunos datos de los que inicialmente no se podía extraer información demasiado útil, por lo que se ha requerido procesarlos y transformarlos hasta tenerlos en un formato adecuado para la construcción de los modelos construidos.

Por otro lado, en cuanto a resolver los problemas planteados en las competiciones en las que se participó, se ha satisfecho en cierta medida, ya que a través de los algoritmos de clasificación empleados se ha podido modelizar los problemas planteados en cada competición. Sin embargo, en la competición del *Titanic* no sabemos la posición final del ranking que se alcanzará, ya que el plazo para esta competición finaliza el 31 de diciembre de 2016. Para la competición *shelter animals* se aplicaron los mismos algoritmos y no se alcanzó una posición muy alta del ranking general.

El impacto que tiene la plataforma *Kaggle* en el campo del aprendizaje automático es muy alto. A los investigadores y a los usuarios principiantes esta plataforma les proporciona experiencia al enfrentarlos a problemas con datos reales, problemas que suponen un reto a la hora de tratar los datos y modelizarlos. Además, en algunas competiciones los usuarios o grupos con mejor resultado obtienen un premio monetario o incluso una oferta de trabajo por parte de la empresa que creó dicha competición.

Por otra parte, a las empresas también les resulta interesante el uso de esta plataforma, ya que a través de la amplia comunidad *Kaggle* y aportando datos propios difíciles de obtener por otras vías, pueden como resultado obtener mayores beneficios, como en el caso de la competición de Bimbo, o mejorar el funcionamiento, como el caso de la competición finalizada hace dos meses por la web yelp, que ofrecía trabajo a quien pudiese implementar un buen algoritmo de etiquetado automático de restaurantes a través de fotos que subidas por los usuarios.

Por mi parte, como usuario principiante en el mundo del aprendizaje automático este trabajo ha sido bastante relevante y satisfactorio, tanto por la experiencia que me ha aportado en cuanto a la programación y utilización de algoritmos capaces de resolver los problemas planteados, como relativa al conocimiento de técnicas de procesamiento de la información. Toda esta experiencia obtenida espero poder utilizarla en un futuro para seguir aplicando las técnicas aprendidas, así como también me gustaría aprender nuevos algoritmos con los que intentar resolver los objetivos de otras competiciones para tratar de alcanzar mejores resultados en los siguientes intentos.

Bibliografía

Aprendizaje Automático. (n.d.). http://www.cs.us.es/~fsancho/?e=75

Árbol de decisión Wikipedia. (n.d.). https://es.wikipedia.org/wiki/Aprendizaje_basado_en_Árboles_de_decisión

Flach, P. A. (2012). *Machine learning: The art and science of algorithms that make sense of data*. Cambridge University Press. http://www.cambridge.org/9781107422223

Grolemund, G. (2014). *Hands-on programming with r: Write your own functions and simulations*. oreilly. http://shop.oreilly.com/product/0636920028574.do

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer. doi:10.1007/978-0-387-84858-7

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling*. Springer. doi:10.1007/978-1-4614-6849-3

LDA Wikipedia. (n.d.). https://es.wikipedia.org/wiki/Análisis_discriminante_lineal#Discriminante_Lineal_de_Fisher

Mediciones del random forest. (n.d.). http://apuntes-r.blogspot.com.es/2015/06/mediciones-del-random-forest.html

Página web de Kaggle. (n.d.). https://www.kaggle.com/

Random forest. (n.d.). http://randomforest2013.blogspot.com.es/2013/05/randomforest-definicion-random-f
html

Wickham, H., & Grolemund, G. (2016). *R for data science: Visualize, model, transform, tidy, and import data*. oreilly. http://shop.oreilly.com/product/0636920034407.do