Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Random Forests para detección de fraude en medios de pago

Trabajo Fin de Máster
Máster en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones

Sara Hidalgo Ruiz-Capillas

bajo la dirección de

Carlos Santa Cruz Fernández

Madrid, 21 de septiembre de 2014

Índice general

Resumen

Los conjuntos de clasificadores constituyen una de las técnicas más eficaces y sencillas para resolver problemas de clasificación. Se basan en la construcción de una serie de clasificadores cuya combinación, bajo unas ciertas condiciones, mejora el resultado obtenido por cada uno de ellos individualmente. Este trabajo se centra en el estudio de por qué y cuáles son las condiciones necesarias para que este tipo de técnicas resulten ser tan eficaces.

El resultado del estudio teórico hace que el algoritmo *random forest* sea elegido como el más adecuado para ser aplicado al problema de clasificación de fraude, el cual se caracteriza por el significativo desbalanceo entre sus clases y el gran volumen de datos manejado. Para ello, se ha desarrollado una librería en C, con la que han sido llevados a cabo experimentos sobre transacciones bancarias reales con dos objetivos principales: por un lado, comparar el rendimiento obtenido por *random forest* con otros algoritmos tanto lineales como no lineales y por otro, analizar el impacto del valor de sus parámetros sobre el rendimiento final del conjunto.

Agradecimientos

Este trabajo es producto del esfuerzo pero también del entusiasmo que ha supuesto para mí el estudiar, conocer y tener la oportunidad de poner en práctica muchos de los conocimientos que he adquirido en este último año, lo cual no hubiera sido posible sin la ayuda de mi tutor. Muchas gracias, Carlos, por tu paciencia, tu dedicación y tu apoyo, y, sobre todo, por tener siempre el consejo adecuado con el que hacerme ver sencillo lo complejo.

También me gustaría agradecer al IIC (Instituto de Ingeniería del Conocimiento) haberme dado la oportunidad de estudiar este máster y haber puesto a mi disposición todos los medios y las facilidades que he necesitado para llevar a cabo este estudio. Muchas gracias en especial a mis compañeros Álvaro Barbero, Alberto Torres y Jesús Prada por sus sugerencias, sus comentarios y su tiempo.

Y por supuesto, a todos los que estáis siempre a mi lado, a mi familia y amigos. En especial, gracias Borja, por ser mi gran compañero, dentro y fuera de las aulas. Por último, me gustaría agradecer a mis padres su apoyo incondicional durante este último año, por haber estado siempre a mi lado en los momentos difíciles y haber sufrido tanto o más que yo tanto mis alegrías como mis preocupaciones. Gracias por haberme inculcado el afán de superación necesario para haber concluido este proyecto.

Capítulo 1

Introducción

La cantidad de información disponible aumenta cada instante. La capacidad de procesamiento y almacenamiento de los datos también. La combinación de ambas realidades hace que el desarrollo de técnicas y métodos de aprendizaje automático para su aplicación sobre grandes cantidades de datos se haya visto muy impulsado en los últimos años y constituya en la actualidad un área muy destacada de estudio.

Aprendizaje automático

Aprendizaje automático es aprender de los datos, es descubrir la estructura y los patrones que subyacen en ellos. El objetivo principal del aprendizaje automático es la extracción de la información contenida en un conjunto de datos con el fin de adquirir conocimiento que permita tomar decisiones sobre nuevos conjuntos de datos. Formalmente, se define como [Mitchell, 2006]:

> Un sistema aprende de la experiencia E con respecto a un conjunto de tareas T y una medida de rendimiento P, si su rendimiento en T, medido según P, mejora con la experiencia E.

Tomemos como ejemplo para ilustrar la definición anterior un sistema de reconocimiento de caracteres. En tal caso, la tarea T es la identificación de palabras a partir de imágenes, la cual puede ser evaluada según P, donde P es la proporción de palabras correctamente identificadas. Para ello, se adquiere conocimiento a partir de la experiencia E, que correspondería a una base de datos de imágenes asociadas a la palabra mostrada.

El reconocimiento de caracteres es sólo una de las muchas aplicaciones del aprendizaje automático. Otros **ejemplos** son:

- Reconocimiento del habla. Todos los sistemas de reconocimiento de voz que existen actualmente en el mercado usan de una forma u otra técnicas y algoritmos

de aprendizaje automático. Se trata de un problema en el que es claro que la precisión obtenida por el sistema es mucho mayor si éste es entrenado para ello que siendo programado explícitamente, lo cual constituye la principal característica de los sistemas basados en técnicas de aprendizaje automático.

- Detección precoz de enfermedades. El sistema parte de historias y registros clínicos ligados a diagnósticos previos correctos con el fin de aprender a identificar futuros pacientes enfermos y servir de apoyo al experto.

- Detección de correo electrónico no deseado (*spam*). Los sistemas capaces de filtrar el correo basura son también sistemas basados en aprendizaje automático que se encargan de clasificar nuevos correos basándose en los que previamente fueron marcados como *spam*. Se trata de un problema que cambia con el tiempo por lo que debe estar preparado para adaptarse a nuevos patrones.

Todos los algoritmos anteriormente descritos se basan en datos que estaban previamente etiquetados (correo *spam*/no *spam*, paciente enfermo/sano) a partir de los cuales se deducen las reglas para clasificar nuevos casos. Este tipo de algoritmos basados en datos etiquetados se enmarcan dentro de lo que se conoce como **aprendizaje supervisado**. A su vez, dichos problemas pueden dividirse en **clasificación** y **regresión**. Ambos se basan en las características (*features*) de cada elemento para predecir su etiqueta de clase pero mientras que en un problema de clasificación esta etiqueta sólo toma valores en un conjunto discreto, lo hace en un conjunto continuo cuando se trata de problemas de regresión.

Otro tipo de problemas son aquellos que se conocen como **aprendizaje no supervisado**. En este caso, los datos no pertenecen a ninguna clase (sin etiquetas) por lo que este tipo de técnicas se centran en la búsqueda de estructuras o distribuciones no conocidas subyacentes en los datos.

Como ya se ha mencionado anteriormente, estos algoritmos se basan en una serie de datos sobre los que aprender para luego aplicar la experiencia adquirida en otros conjuntos; de nada sirve evaluar la capacidad predictiva de un sistema sobre el conjunto de datos que el algoritmo ya ha visto. Por tanto, es necesario evaluar su rendimiento sobre un conjunto distinto a aquél con el que el sistema fue entrenado para obtener una estimación válida de su capacidad de generalización ante nuevos ejemplos. Con este fin, el conjunto de datos disponible se divide en dos subconjuntos: **conjunto de entrenamiento** y **conjunto de test**. De esta forma, el modelo es generado a partir de los datos de entrenamiento y evaluado sobre el conjunto de test, sobre el cual se puede medir la precisión del modelo ya que disponemos de las etiquetas de cada elemento. El resultado obtenido sobre este conjunto es una buena aproximación al que se espera obtener para datos nuevos.

Por tanto, la generalización es uno de los aspectos clave en el diseño de algoritmos de aprendizaje automático pero a la vez los modelos deben ajustarse al conjunto de entrenamiento y captar toda su información. Surge así el problema de la compensación **sesgo-varianza** (sección 2.2); el sesgo mide el error medio del modelo utilizando

distintos conjuntos de entrenamiento mientras que la varianza mide la sensibilidad del modelo a pequeños cambios en los datos de entrenamiento. Es decir, modelos muy complejos como pueden ser polinomios de alto grado tienen sesgo bajo (representaciones muy aproximadas del conjunto de entrenamiento) y varianza alta (gran variabilidad ante los cambios en el conjunto de entrenamiento), lo cual se conoce también mediante el término *overfitting* (sobreajuste). Por otro lado, modelos simples como un modelo lineal tienen sesgo alto pero varianza muy baja.

Pese a ser pasos fundamentales en el desarrollo de un sistema basado en técnicas de aprendizaje automático, la construcción y evaluación de los modelos sólo constituye una parte de todo el proceso de construcción del sistema. El proceso de diseño puede dividirse en [Duda et al., 2001]:

1. **Recolección de los datos.** Suele constituir un proceso tedioso que ocupa gran parte del desarrollo del sistema ya que generalmente se hace necesario recopilar grandes cantidades de datos para poder asegurar que la muestra usada es representativa del conjunto estudiado.

2. **Elección de características.** Se trata de un paso crítico ya que es necesario extraer aquellas variables que sean útiles para distinguir los patrones de cada categoría. En este paso, el conocimiento experto puede resultar útil.

3. **Elección del modelo.** En este paso se elegirá el modelo que más se ajuste a nuestro problema y que consiga el rendimiento esperado sobre el conjunto de test. Este modelo, entre otras cosas, deberá mantener el equilibrio sesgo-varianza explicado anteriormente.

4. **Entrenamiento del modelo.** En esta fase se construye el clasificador, cuyos parámetros se ajustan a partir del conjunto de datos de entrenamiento. Encontrar los parámetros que se ajusten a nuestro modelo constituye un problema de optimización ya que el objetivo es siempre minimizar una cierta función objetivo.

5. **Evaluación del modelo.** Haciendo uso de un conjunto de datos independiente del que se usó para entrenar, se fija una medida de error y se obtiene el rendimiento del modelo. Si el resultado no es el esperado, se deberá probar retrocediendo a cada uno de los puntos anteriores y retomar el proceso de nuevo.

Big Data

En los últimos años, el término *Big Data* se ha convertido en uno de los más utilizados en el área de las tecnologías de la información y la comunicación (TIC). Aunque no existe una definición precisa y cuantitativa de lo qué es o no *Big Data*, se considera *Big Data* a todo conjunto de datos cuyo tamaño no puede ser procesado en tiempo razonable con la infraestructura hardware y las herramientas software comúnmente utilizadas. Dadas estas limitaciones, infraestructuras, tecnologías y servicios han sido desarrollados para soportar el manejo de estas grandes bases de datos.

Este escenario puede ser ilustrado mediante una pirámide cuya base sería la infraestructura hardware necesaria, sobre la cual se sustentan las tecnologías y servicios que sirven de soporte al fin último del *Big Data*, la aplicación de algoritmos sobre los datos y la extracción de conocimiento a partir de ellos (figura 1.1).

Figura 1.1: *Big Data*

De forma un poco más concreta, el *Big Data* se caracteriza y describe mediante lo que se conoce como sus "3Vs"[Beyer and Laney, 2012]:

- **Volumen.** Disponer de grandes volúmenes de datos y poder procesarlos constituye uno de los mayores atractivos del *Big Data* ya que la efectividad de los algoritmos que se apliquen sobre ellos está directamente relacionada con su tamaño. Arquitecturas escalables y procesamiento en paralelo de la información son necesarios para lidiar con esta primera V del *Big Data*.
- **Velocidad.** Velocidad hace referencia a la rapidez con la que los datos se reciben, se procesan y se toman decisiones a partir de ellos. Cada vez en más sistemas, tan importante como obtener la respuesta correcta es obtenerla a tiempo.
- **Variedad.** La tercera V hace referencia a la diversidad en las fuentes de los datos. La información disponible suele ser muy diversa y no estructurada haciéndose necesario un cuidadoso preproceso de los datos antes de poder utilizarlos y obtener utilidad de ellos.

Sin embargo, en base a la experiencia, se suele añadir otras dos uves a la definición anterior: **Veracidad y Valor**.

La veracidad se refiere a fiabilidad. Pese a que el *Big Data* permite una mayor flexibilidad que las bases de datos tradicionales en cuanto a la consistencia de los datos, partir de información fiable es clave para obtener los resultados deseados y tomar las decisiones correctas.

El valor está estrechamente ligado con la utilidad; descubrir el valor de los datos consiste en extraer la información que subyace en ellos, en encontrar las relaciones implícitas y explícitas entre los datos y obtener conocimiento que, de una u otra forma, aporte calidad y mejore aquello de lo que partíamos previamente.

Aprender del *Big Data*. La detección de fraude.

El objeto de este trabajo es la detección de fraude en medios de pago, problema que se contextualiza dentro de las dos áreas previamente descritas. Partiendo de todas las transacciones bancarias realizadas por los clientes de un determinado banco, el objetivo es la clasificación de operaciones futuras detectando el mayor número posible de casos de fraude y minimizando los falsos positivos en las transacciones genuinas. Una operación se considera fraudulenta cuando no es el cliente legítimo el que realiza la transacción, bien porque su tarjeta ha sido robada o clonada o bien porque sus credenciales de acceso al banco por Internet han sido comprometidos.

Dentro del área del **aprendizaje automático**, se trata de un problema de clasificación de dos clases (fraude/no fraude) que parte de un conjunto de datos etiquetados y que, por tanto, se enmarca dentro del aprendizaje supervisado. En cuanto a las distintas fases de desarrollo de un sistema de aprendizaje automático, se presuponen fijas las fases previas a la construcción del modelo (recolección de datos, extracción de características y codificación), quedando todas ellas fuera del ámbito de este estudio.

Su relación con el ***Big Data*** es también inmediata: el Volumen y la Velocidad son claves en un sistema de detección de fraude. Millones de operaciones con tarjeta y cientos de miles de transferencias bancarias son realizadas diariamente en una sola entidad bancaria, las cuales deben ser procesadas en ventanas de pocos milisegundos para poder tomar una decisión en tiempo real. El valor es otro aspecto determinante ya que más allá de ser un interesante y complejo problema de clasificación a nivel teórico, el uso de un sistema de detección de fraude supone el ahorro de un porcentaje importante de las pérdidas que el fraude supone a grandes bancos en todo el mundo.

La presente memoria describe el estudio realizado en torno a dicho problema de clasificación centrándose en el algoritmo *random forest*, uno de los muchos métodos basados en **conjuntos de clasificadores**. Se divide en los siguientes capítulos:

- En el capítulo 2 se incluye una revisión de los conceptos centrales en los que se basa el algoritmo *Random Forest*: los árboles de decisión y los conjuntos de clasificadores.
- El capítulo 3 describe las características y peculiaridades del problema de detección del fraude.
- En el capítulo 4 se aplican las técnicas descritas en el capítulo 2 al problema de la detección del fraude y se exponen los resultados obtenidos.
- El capítulo 5 recoge las conclusiones obtenidas y las posibles líneas de trabajo futuro.

Capítulo 2

Árboles de decisión y Conjuntos de Clasificadores

En un problema de clasificación supervisada se dispone de una muestra $\mathcal{L}$ que se define como

$$\mathcal{L} = \{(\mathbf{x_n}, y_n) \mid n = 1, 2, \ldots, N, \mathbf{x_n} \in \mathbb{R}^d, y_n \in \{1, 2, \ldots, C\}\} \ , \tag{2.1}$$

siendo N el número de elementos del conjunto de datos, C el número de clases distintas y d el número de variables que definen los ejemplos $\mathbf{x_n}$ del conjunto. Cada uno de estos ejemplos se representa mediante un vector $\mathbf{x_n}$, el cual tiene asociada su correspondiente etiqueta de clase y_n y está definido por distintas variables, las cuales pueden ser numéricas (sus valores son números reales) o categóricas (toman valores en un conjunto finito en el que no existe ninguna relación de orden). Dichos vectores $\mathbf{x_n}$ se conocen también como vectores de características.

Partiendo de estos datos, un algoritmo de aprendizaje trata de encontrar una función $f : \mathbb{R}^d \to \{1, 2, \ldots, C\}$ que sea capaz de predecir la clase de nuevos elementos y cuyo error de clasificación sea pequeño, es decir, $\#\{n \mid f(\mathbf{x_n}) \neq y_n\} \simeq 0$. Esta función objetivo f es un elemento de una familia de funciones $\mathcal{F}$, a la cual se denomina *espacio de hipótesis*. Por tanto, para resolver un problema de clasificación se debe responder a dos cuestiones: cuál sera la familia de funciones $\mathcal{F}$ elegida y qué valores deben tener los parámetros de f para que se ajusten lo más posible al conjunto de datos $\mathcal{L}$.

La respuesta a la primera cuestión debe encontrarse en base a las características del problema y al conocimiento experto mientras que son los **algoritmos de aprendizaje** los que nos permiten ajustar y optimizar los parámetros del clasificador elegido. Algunos de los modelos más importantes dentro de la clasificación supervisada son: Naive-Bayes, discriminantes lineales (LDA), árboles de decisión, redes neuronales y máquinas de vectores soporte ([Duda et al., 2001; Mitchell, 2006]). A partir de estos clasificadores, se han desarrollado técnicas que se basan en la combinación de varios de ellos y en los que la decisión final se toma a partir de las decisiones individuales de cada uno ([Breiman, 1996a; Dietterich, 2000a; Freund and Schapire, 1995]). Su objetivo es obtener una clasificación más precisa que la que obtendríamos si los tomásemos por separado.

El objeto de estudio en la parte experimental de este trabajo es el algoritmo ***random forest*** [Breiman, 2001], el cual está formado por un conjunto de árboles de decisión. En este capítulo se describen los dos conceptos sobre los que se fundamenta: los árboles de decisión y los conjuntos de clasificadores.

2.1. Árboles de decisión

Un árbol de decisión T es una secuencia ordenada de preguntas en las que la siguiente pregunta depende de la respuesta a la pregunta actual. Dichas cuestiones son formuladas sobre las variables que definen cada elemento $\mathbf{x}$ con el fin de acabar asignándoles una determinada clase y. Este procedimiento, con sus correspondientes preguntas y bifurcaciones, es representado de forma natural mediante un árbol.

En un árbol de decisión el primer nodo se conoce como nodo raíz, el cual está conectado sucesivamente con el resto de nodos hasta alcanzar los nodos hoja, aquellos que no tienen descendientes. A cada nodo interno se le asigna una de las preguntas de la secuencia mientras que a cada nodo hoja le es asignada una etiqueta de clase. De esta forma, la pregunta del nodo raíz es formulada a todo el conjunto $\mathcal{L}$, el cual se va subdividiendo hasta alcanzar los nodos hoja, que constituyen una partición disjunta del espacio de características inicial. Esto ocurre porque dado un nodo, una y sólo una rama será seguida por cada ejemplo $\mathbf{x}$ del conjunto de entrenamiento.

Antes de describir los métodos de construcción de árboles, se ilustrará el procedimiento de clasificación mediante un ejemplo.

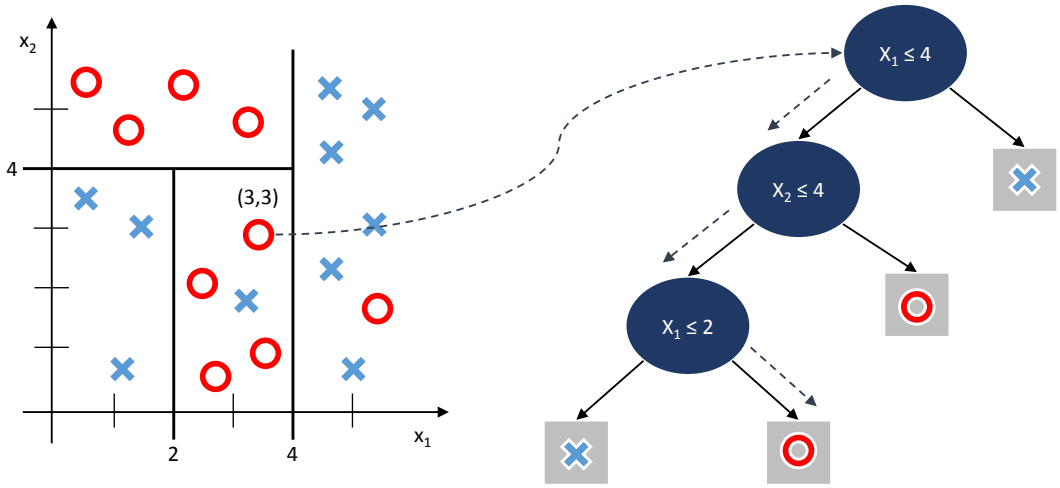

Figura 2.1: Ejemplo de árbol de decisión

En la figura 2.1 se ha representado un conjunto de datos en $\mathbb{R}^2$ en el que distinguimos dos clases. Un posible árbol de decisión para dicho problema de clasificación es el que encontramos a su derecha, el cual define las cuatro regiones marcadas en el espacio de características. En este caso, sólo dos de todos los ejemplos representados

serían incorrectamente clasificados por el árbol de la derecha, en el que se ilustra la clasificación de uno de los datos cuya etiqueta se predice correctamente 2.1.

Como podemos observar, el árbol anterior divide el espacio de características mediante hiperplanos paralelos a los ejes, lo que significa que cada una de las preguntas asignadas a los nodos hace referencia a una sola variable (son del tipo ¿$x_n \leq \alpha$?). Pese a que divisiones oblicuas son también usadas para ciertos problemas, no serán tratadas en este estudio.

Una vez explicados los fundamentos de los árboles de decisión, es interesante remarcar dos grandes ventajas respecto de otros algoritmos de clasificación:

- Permiten incluir variables categóricas dentro del conjunto de atributos de los elementos **x**. En este caso, a la hora de realizar las preguntas de los nodos, en vez de usar relaciones de orden, se utilizan subconjuntos de valores dentro de todos los posibles para una determinada variable. No serán incluidas más referencias a este tipo de atributos ya que su tratamiento queda fuera del ámbito de este estudio.

- Pueden ser interpretados y entendidos como conjuntos de reglas (del tipo `if` $[x_1 > 2$ `and` $x_3 < 8]$ `then` $x \in y_1)$, lo cual hace que las decisiones tomadas a partir de ellos puedan ser justificadas. Otros algoritmos de aprendizaje, como las redes neuronales, actúan como cajas negras en las que la decisión tomada es difícilmente explicable.

A continuación, en la sección 2.1.1 se describirá el algoritmo CART [Breiman et al., 1984] y en la sección 2.1.2 se esbozarán las particularidades de otros algoritmos de construcción de árboles.

2.1.1. Árboles CART

El procedimiento de construcción de un árbol CART (Classification and Regression Trees) parte del conjunto total de datos y sobre él define el mejor corte posible dividiendo el conjunto inicial en dos subconjuntos. Sobre ellos y recursivamente, se obtendrán los sucesivos cortes que definirán el árbol. Por tanto, los tres problemas a resolver son:

1. Seleccionar la variable y el valor de corte en cada nodo.

2. Asignar una clase a cada nodo hoja.

3. Decidir el criterio para definir un nodo como hoja o continuar construyendo el árbol.

A continuación se detalla cómo el algoritmo CART resuelve estos puntos.

Selección de la variable y el valor de corte.

El objetivo fundamental al seleccionar el corte en un árbol de decisión es que dado un nodo t, su hijo izquierdo t_L y derecho t_R sean lo más *puros* posible con respecto a su padre. Es decir, que las clases estén siempre mejor separadas en los descendientes que en los progenitores. Antes de explicar cómo se obtiene este valor de corte, es importante remarcar que CART se centra en la construcción de **árboles binarios** (cada nodo tiene o dos hijos o ninguno) debido a su simplicidad y a que todo árbol con más ramas en alguno de sus nodos puede ser reducido a un árbol binario aumentando su profundidad. De la misma manera, se particularizará de ahora en adelante a problemas de clasificación de **dos clases**, ya que el problema abordado en este trabajo, la clasificación de fraude, es un problema de este tipo (capítulo 3).

Por tanto, se debe definir una **medida de la impureza** para cada nodo $i(t)$, la cual debe ser máxima cuando las clases estén presentes en t en la misma proporción (nodo menos *puro* posible) y ser 0 cuando sólo haya elementos de una de las clases (nodo más *puro* posible). Formalmente, una función de impureza es una función ϕ definida sobre (p_1, p_2) tal que $p_1, p_2 \geq 0$ y $p_1 + p_2 = 1$ y cumple las siguientes propiedades:

1. ϕ alcanza su máximo en el punto $\left(\frac{1}{2}, \frac{1}{2}\right)$
2. ϕ alcanza su mínimo en los puntos $(1, 0), (0, 1)$.
3. ϕ es una función simétrica.

Dada una función de impureza ϕ podemos definir entonces la impureza de un nodo t como

$$i(t) = \phi(p(1|t), p(2|t)) \ , \tag{2.2}$$

donde $p(y|t)$ es la probabilidad de pertenecer a la clase y dado el nodo t y se calcula (en el caso de dos clases) de la siguiente forma:

$$p(1|t) = \frac{N_1(t)}{N(t)} \ , \ p(2|t) = \frac{N_2(t)}{N(t)} = 1 - \frac{N_1(t)}{N(t)} \ ,$$

donde $N(t)$ son el número de ejemplos del nodo t y $N_1(t)$ y $N_2(t)$ los pertenecientes a la clase 1 y 2, respectivamente. Se cumple, por tanto, que $N(t) = N_1(t) + N_2(t)$

La función de impureza utilizada por el algoritmo CART (por ser la más simple en el caso de dos clases) es la impureza **GINI**, la cual cumple las propiedades anteriores y se define como

$$i(t) = p(1|t) \cdot p(2|t) \ . \tag{2.3}$$

Por tanto, se puede también expresar como

$$i(t) = \frac{N_1(t)}{N(t)} \cdot \left(1 - \frac{N_1(t)}{N(t)}\right) = p(1|t) \cdot (1 - p(1|t)) \ . \tag{2.4}$$

La impureza GINI no es la única medida de impureza. Existen también otras como la entropía:

$$i(t) = -(p(1|t) \cdot \log_2 p(1|t) + p(2|t) \cdot \log_2 p(2|t)) \ , \tag{2.5}$$

o el error de clasificación:

$$i(t) = min\{p(1|t), p(2|t)\} \ . \tag{2.6}$$

Aunque existen distintas funciones de impureza, se ha observado que la elección de una u otra no es muy determinante en cuanto a la capacidad de generalización del árbol final [Breiman et al., 1984]. En la figura 2.2 se observan las tres medidas para el caso de dos clases ajustadas en escala para facilitar la comparación.

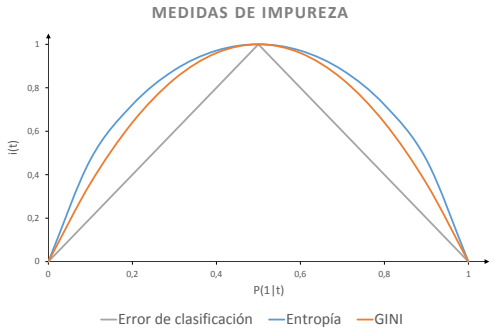

Figura 2.2: Comparación entre funciones de impureza

Una vez definida la función de impureza, el corte óptimo será aquel que maximice la diferencia entre la impureza del nodo padre y la de sus hijos, teniendo en cuenta la proporción de datos que van a cada uno de ellos. Es decir, el corte óptimo será aquel que maximice

$$\Delta i(t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R) \ , \tag{2.7}$$

donde $i(t_L)$ y $i(t_R)$ son las impurezas del hijo izquierdo y derecho y p_L y p_R la proporción de datos de t que van hacia cada uno de los nodos hijos.

La forma de obtener este valor óptimo es realizar una búsqueda exhaustiva en todas las variables posibles con todos los posibles cortes. Los valores *candidatos* son todos aquellos que dada una variable x_i conducen a distintas divisiones de los datos. Supongamos que la variable x_i tiene V valores distintos que ordenados se pueden representar como $\{u_1, u_2, \ldots, u_V\}$. Para esta variable x_i se evaluará la diferencia de impureza (ec. 2.7) $V-1$ veces (para $x_i \leq u_1?, \ldots, x_i \leq u_{V-1}?$) y se elegirá el valor que la maximice. CART fija el valor de corte en $(u_i + u_{i+1})/2$ si el valor máximo se alcanza en u_i (se toma el punto medio ya que cualquier valor en el intervalo $[u_i, u_{i+1})$ supone la misma separación y por tanto la misma reducción de impureza). Dicho procedimiento será repetido para cada variable y se tomará el valor máximo entre todas ellas.

La búsqueda en las variables se realizará en cada uno de los nodos hasta que o bien todos los nodos contengan elementos de una sola de las clases (impureza 0) o bien se cumpla un criterio de parada.

Asignación de clase a los nodos hoja.

Una vez se ha decidido que el nodo actual t debe pertenecer al conjunto de nodos terminales $\tilde{T}$, asignarle una clase es inmediato. Si el árbol ha sido construido hasta que los nodos hoja contienen elementos de una sola clase (impureza 0), la clase a la que pertenecen todos los elementos será la asignada al nodo. Si el nodo no es puro, bien porque ha sido aplicada poda (*pruning*) o bien porque se utilizó algún criterio de parada, la clase asignada será aquella a la que corresponda la mayoría de los ejemplos del nodo. En caso de que ambas clases estén presentes en el nodo en la misma proporción, la etiqueta se asignará de forma arbitraria. Esto es

$$t \text{ es etiquetado como } y \;\Leftrightarrow\; y = \arg\max_{y} p(y|t) \; . \tag{2.8}$$

Criterios de parada. Poda.

Los árboles de decisión construidos hasta que todos los nodos tienen impureza 0 han resultado ser muy ineficientes, ya que no tienen capacidad de generalización. Esto ocurre porque aunque todos los patrones están bien clasificados en el conjunto de entrenamiento, los errores de clasificación en el conjunto de test suelen ser elevados (*overfitting*). Esta situación puede ser resuelta de dos formas: utilizando criterios de parada (prepoda) o podando el árbol una vez ha sido construido hasta el final (post-poda, denotada de ahora en adelante simplemente como poda).

La primera aproximación consiste en utilizar una **regla de parada** en la que fijado un cierto umbral β, un nodo t se considera terminal si el mejor corte posible a partir de él cumple que $\Delta i(t) < \beta$. De acuerdo con [Breiman et al., 1984], los resultados obtenidos con este tipo de reglas, nunca han conseguido ser satisfactorios. Por un lado, si el valor de β es demasiado pequeño, el problema es el mismo que para el árbol construido hasta el final: árboles demasiado grandes sobreajustados al conjunto de entrenamiento. Por otro lado, incrementar el valor de β puede conducir a que la construcción del árbol se pare en un cierto nodo con el máximo $\Delta i(t)$ pequeño pero cuyos descendientes t_L y t_R podrían haber conducido a descensos de impureza significativos. Por tanto, optimizar el criterio de parada no parece la forma correcta de atacar el problema mientras que realizar una poda al árbol una vez construido hasta el final sí parece el enfoque correcto para conseguir que el árbol tenga capacidad de generalización [Breiman et al., 1984].

Para aplicar el **procedimiento de poda**, se debe comenzar definiendo el estimador del error local de clasificación $R(t)$ para un nodo t:

$$R(t) = r(t) \cdot p(t) \; , \tag{2.9}$$

donde $r(t) = min\{p(1|t), p(2|t)\}$ o, lo que es lo mismo, la probabilidad de que un elemento sea mal clasificado y $p(t) = N(t)/N$ con N el número total de ejemplos del entrenamiento y $N(t)$ el número de elementos de este mismo conjunto que pertenecen al nodo t. Cuando los árboles son construidos según el método anterior, siempre se cumple que

$$R(t) \geq R(t_L) + R(t_R) \; . \tag{2.10}$$

A partir de los errores locales $R(t)$ se define el error global $R(T)$:

$$R(T) = \sum_{t \in \tilde{T}} R(t) \ , \tag{2.11}$$

donde $\tilde{T}$ hace referencia al conjunto de nodos terminales de T. La idea intuitiva de este error es muy clara ya que corresponde a la suma en todos los nodos hoja de la probabilidad de caer en un determinado $t \in \tilde{T}$ multiplicado por la probabilidad de ser mal clasificado en dicho nodo.

En el caso de los árboles construidos hasta el final, $R(T)$ es 0 para el conjunto de entrenamiento, lo cual constituye un estimador demasiado optimista del error de clasificación esperado para nuevos ejemplos (al que denotaremos como $R^*(T)$). Esto ocurre porque mientras que $R(T)$ siempre disminuye para el conjunto de entrenamiento según avanza la construcción del árbol, $R^*(T)$ disminuye hasta alcanzar un mínimo a una cierta profundidad pero a partir de ese punto empieza a aumentar. Dicho mínimo es el que se espera alcanzar tras aplicar la poda al árbol ya que es en ese punto donde se consigue el equilibrio sesgo-varianza. De la misma forma, mediante la poda, se conseguirá que $R(T)$ constituya un mejor estimador del error esperado $R^*(T)$.

El primer paso consiste en construir un árbol T_{max} a partir de todo el conjunto de datos de entrenamiento de tal forma que los nodos terminales contengan elementos de una sola clase. Una vez construido, CART utiliza un criterio de poda denominado **criterio de coste-complejidad**, el cual selecciona un sub-árbol de entre todos los contenidos en T_{max} (un sub-árbol T de T_{max} se denota como $T \preceq T_{max}$). Dicho criterio trata de buscar un equilibrio entre coste y complejidad en el sentido de que dados dos árboles con el mismo error, será preferible el de menor complejidad mientras que entre dos árboles del mismo tamaño (complejidad) se optará por el que tenga menor error. Por tanto, la medida coste-complejidad se define para cada $T \preceq T_{max}$ como

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}| \ , \tag{2.12}$$

donde α es el parámetro de complejidad ($\alpha \geq 0$) y $|\tilde{T}|$ el número de nodos terminales de T.

El objetivo es encontrar, para cada valor de α, el árbol $T(\alpha)$ que minimiza la ecuación anterior (ec. 2.12). Cuando $\alpha = 0$, la complejidad del árbol no es penalizada por lo que $T(0) = T_{max}$ (ya que $R(T_{max}) = 0$), mientras que al aumentar el valor de α, los árboles $T(\alpha)$ tendrán cada vez menos nodos terminales (la complejidad empieza a penalizarse más). El caso extremo se da para un cierto valor α_K en el que la penalización por la complejidad prevalece totalmente frente al error $R(T)$ y el árbol elegido será sólo el nodo raíz. Por tanto, existe una secuencia finita de sub-árboles obtenida a partir de los distintos valores de α en los que el árbol minimizador cambia ($\alpha_0 = 0, \alpha_1, \ldots, \alpha_K$):

$$T_{max} = T(0) \succeq T(\alpha_1) \succeq \cdots T(\alpha_K) = \text{raiz}(T) \ .$$

El procedimiento para obtener dicha secuencia es el siguiente:

1. Obtener $T(\alpha_1)$. El árbol de partida $T(\alpha_1)$ debe ser el árbol más pequeño posible cuyo error global de clasificación es el mismo que el de T_{max}. En caso de haber construido T_{max} hasta que los nodos terminales tienen sólo elementos de una clase, el propio T_{max} es ya el árbol más pequeño posible ya que todos sus nodos hoja implicaron un descenso de impureza respecto de la de sus nodos padre (todos sus nodos hoja tienen impureza 0 mientras que no era así para sus padres ya que se hubiera dejado de construir en ellos). Por tanto, es imposible reducirlo y a la vez mantener su error $R(T_{max})$.

   En el caso de que T_{max} haya sido obtenido mediante otro procedimiento (criterio de parada, sólo un ejemplo en los nodos terminales, etc.) se debe podar todo par de nodos terminales t_L y t_R que compartan como padre inmediato a t y cumplan $R(t) = R(t_L) + R(t_R)$ hasta que la igualdad no se cumpla para ningún par de nodos terminales.

   De esta forma, $T(\alpha_1)$ cumple que $R(t) > R(T_t)$ (nótese la desigualdad estricta) para todo nodo no terminal t donde T_t es el sub-árbol que tiene t como nodo raíz y contiene todos los descendientes de t en T.

2. Obtener α_2 y el sub-árbol $T(\alpha_2)$ asociado. Para todo nodo no terminal $t \in T(\alpha_1)$ podemos definir dos medidas de coste-complejidad:

   Asociada al nodo individual t: $R_\alpha(t) = R(t) + \alpha$ (con $R(t)$ según ec. 2.9)
   Asociada al sub-árbol T_t: $R_\alpha(T_t) = R(T_t) + \alpha|\tilde{T}_t|$ (con $R(T_t)$ según ec. 2.11)

   Para $\alpha = 0$ se cumple $R_\alpha(t) > R_\alpha(T_t)$ (por el paso 1) pero existe un cierto valor de α en el que la igualdad $R_\alpha(t) = R_\alpha(T_t)$ se cumple:

$$R(t) + \alpha = R(T_t) + \alpha|\tilde{T}_t| \Rightarrow \alpha = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1} \ . \tag{2.13}$$

   Debido a que la poda coste-complejidad tiene como objetivo ir eliminando primero las ramas más débiles, el nodo t_1 cuyos hijos se podarán en primer lugar será aquel que minimice la expresión anterior (ec. 2.13). Aquel nodo para el que la igualdad se satisface con un α más pequeño es aquel en el que la diferencia entre el error del nodo t_1 y del árbol T_{t_1} es más pequeña (ponderada con la complejidad). De esta forma:

$$T(\alpha_2) = T(\alpha_1) - T_{t_1} \ \text{ donde } \ \alpha_2 = \min_{t \in T(\alpha_1)} \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1} \ . \tag{2.14}$$

   El procedimiento de poda $T(\alpha_1) - T_{t_1}$ se ilustra en la figura 2.3.

3. Repetir el paso 2 hasta que el valor de α sea α_K con $T(\alpha_K) = \text{raiz}(T)$.

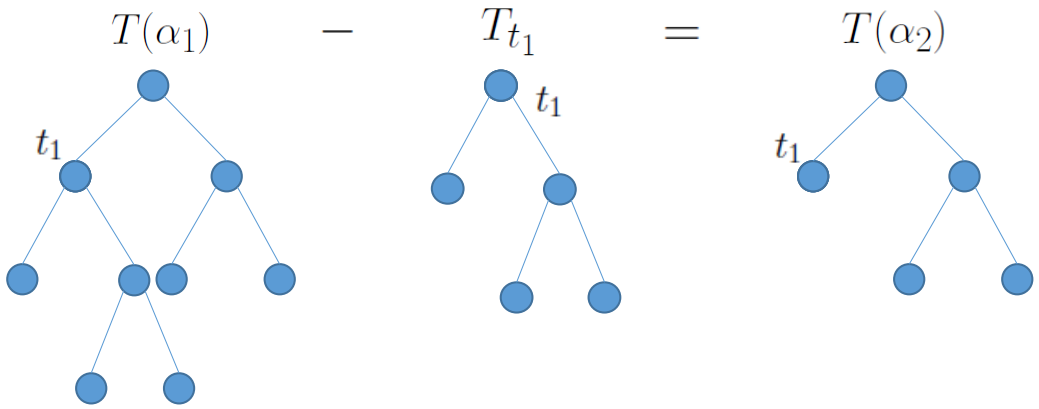

Figura 2.3: Poda del sub-árbol T_{t_1} a $T(\alpha_1)$

Una vez han sido obtenidos todos los valores de α y sus árboles $T(\alpha)$ correspondientes, se debe elegir el árbol cuyo tamaño sea óptimo en el sentido de que su error de clasificación para nuevos ejemplos sea lo más pequeño posible. En este caso, como ya se ha señalado anteriormente, el estimador $R(T)$ no es válido ya que T_{max} sería siempre el árbol seleccionado y de éste sabemos que está sobreajustado al conjunto de entrenamiento. Por tanto, es necesario obtener un mejor estimador de $R^*(T)$ (error real de clasificación), lo cual se puede conseguir de dos formas: usando un conjunto de validación independiente del de entrenamiento o mediante validación cruzada.

Para crear un conjunto de validación, N_v elementos deben ser seleccionados de entre los N datos del conjunto original $\mathcal{L}$. El árbol T_{max} y la secuencia $T(0) \succeq T(\alpha_1) \succeq \cdots T(\alpha_K)$ se construyen únicamente usando los $N - N_v$ datos restantes. Una vez se ha obtenido la secuencia anterior, los N_v datos se clasifican en cada uno de los árboles de la secuencia. El último paso es obtener para cada uno de los árboles la proporción de casos que ha clasificado mal y seleccionar como árbol óptimo aquel con menor error de clasificación. El error obtenido en dicho árbol sí constituye además un buen estimador de $R^*(T)$ ya que los ejemplos que ha clasificado no habían sido vistos antes por el árbol.

La estimación del tamaño óptimo del árbol mediante validación cruzada se realiza dividiendo el conjunto original de entrenamiento $\mathcal{L}$ en V conjuntos disjuntos de tal forma que se construyen V árboles originales T_{max} y V secuencias de árboles $T \preceq T_{max}$ usando los datos contenidos en $V - 1$ conjuntos, por lo que a cada uno le sobra un conjunto para calcular el α óptimo. A partir de los errores obtenidos para cada valor de α en cada conjunto de validación, se obtiene el valor de α que se espera óptimo para el árbol construido con todo el conjunto $\mathcal{L}$. Los detalles de este procedimiento no son incluidos en esta memoria ya que las técnicas de validación cruzada son utilizadas en problemas en los que el número de datos es pequeño y todos deben ser aprovechados para la construcción del modelo, que no corresponde al caso del problema de clasificación de fraude.

2.1.2. Otros métodos

Todos los métodos que desarrollan árboles de decisión se basan en las técnicas anteriormente descritas para el algoritmo CART y sólo difieren de éste en algunos detalles. A continuación se nombran y describen de forma muy general los otros métodos más extendidos y sus principales diferencias con lo anteriormente expuesto:

- **ID3.** Es un algoritmo enfocado al uso de variables categóricas. Si existen variables reales, se agrupan primero sus valores en intervalos para luego ser tratadas como variables categóricas (sin relación de orden). Otra de sus características distintivas es que sólo considera como variables posibles para la división del nodo aquellas que no han sido utilizadas anteriormente y por lo tanto, los árboles generados mediante este método tienen como máximo tanta profundidad como tenga la dimensión de los datos [Duda et al., 2001].
- **C4.5.** Es el sucesor de ID3 y el otro método más utilizado para la generación de árboles de decisión junto con CART. Algunas de las diferencias respecto de CART son: permite nodos con más de dos hijos, el valor usado para el corte en un nodo no corresponde al punto medio entre dos valores de una variable sino que usa el valor de la menor y se basa en la entropía para calcular la impureza. El procedimiento de poda es distinto del usado por CART ya que utiliza heurísticas basadas en test de significancia estadística. La diferencia entre los resultados finales obtenidos con ambas podas es que mientras que la poda CART tiende a generar árboles más pequeños que el óptimo, la poda C4.5 tiende a podar menos de lo necesario [Duda et al., 2001].

2.2. Conjuntos de clasificadores

Los conjuntos de clasificadores (*ensembles*) son sistemas en los que los resultados obtenidos por una serie de clasificadores individuales (clasificadores base) son combinados para predecir las etiquetas de nuevos ejemplos, con el fin de que la precisión del conjunto sea mayor que la obtenida por cada uno de los clasificadores de forma individual. A nivel general, el rendimiento obtenido por el conjunto será mayor que el obtenido por cada uno de sus elementos bajo las siguientes condiciones:

- **Precisión.** El error cometido por cada uno de los clasificadores es menor que el obtenido si los elementos fueran clasificados aleatoriamente ($error < 0{,}5$).
- **Diversidad.** Los errores cometidos por los clasificadores individuales no están correlacionados (patrones incorrectamente clasificados por algunos de ellos deben ser correctamente clasificados por otros). La diversidad es fundamental para mejorar el rendimiento de los clasificadores base ya que la combinación de clasificadores que cometiesen los mismos errores no implicaría ninguna mejora de rendimiento respecto del obtenido por cada uno de ellos de forma individual.

Para demostrar como la combinación de ambas condiciones hace que el error del conjunto sea mucho menor que el de las predicciones de los clasificadores individuales, consideremos el caso simplificado de un conjunto de T clasificadores con errores independientes donde todos ellos tienen la misma probabilidad de acierto, p, y donde el resultado final del conjunto de clasificadores C es la clase votada mayoritariamente. En este caso, la probabilidad de que el conjunto de clasificadores etiquete mal un ejemplo viene dada por la función de distribución acumulada de una distribución binomial, donde menos de $T/2$ clasificadores aciertan. La distribución binomial representa la probabilidad de obtener un número determinado de aciertos en una secuencia de T eventos independientes con probabilidad de éxito p, por lo que el error del conjunto de clasificadores anterior se puede obtener de la siguiente forma:

$$err_C = P(X \leq T/2) = \sum_{i=0}^{\lfloor T/2 \rfloor} \binom{T}{i} p^i (1-p)^{T-i} \, . \tag{2.15}$$

La gráfica 2.4 muestra el error para un conjunto de clasificadores en función de su número de elementos para distintos valores de p (probabilidad de acierto del clasificador base). Se observa que el error decrece según aumenta el número de clasificadores sólo en el caso de que el error de los clasificadores base sea menor de 0,5 (en caso contrario, para clasificadores base con error mayor que un modelo aleatorio, el error aumenta al aumentar el tamaño del conjunto). También se puede ver que, si el error es menor de 0,5, cuanto mayor es el error en los clasificadores base, un mayor número de clasificadores es necesario para que el error global se aproxime a 0.

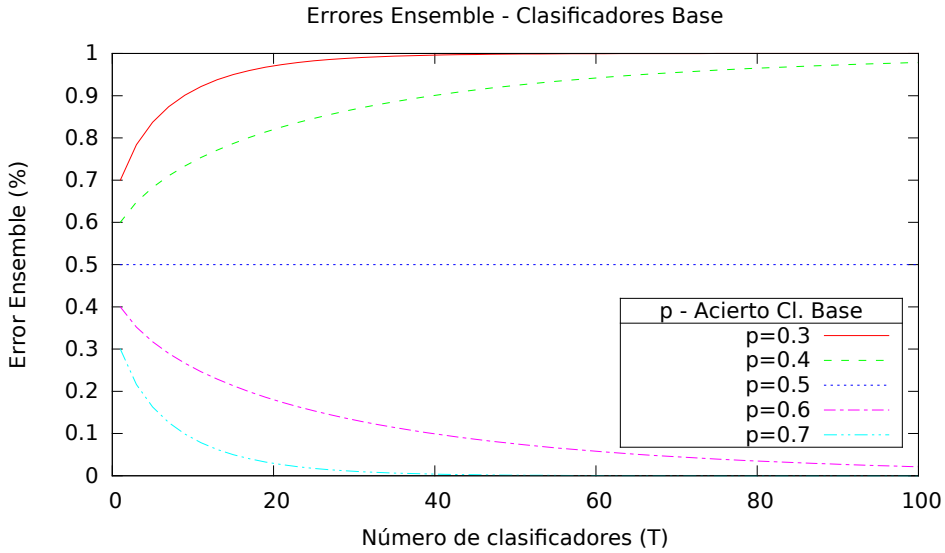

Figura 2.4: Errores Ensembles - Clasificadores Base

Por tanto, el punto clave para obtener conjuntos de clasificadores cuyo error tienda a 0 es conseguir que todos los clasificadores base sean lo más diferentes posible (sus errores no estén correlacionados) sin empeorar demasiado su rendimiento. Con este fin, existen

distintas **técnicas para aumentar la diversidad** entre los distintos clasificadores de un conjunto [Dietterich, 2000a]:

- **Manipulación en los datos de entrenamiento.** El algoritmo de clasificación elegido es ejecutado sobre distintos subconjuntos de los datos de entrenamiento para generar los distintos clasificadores base. Esta técnica funciona con los llamados algoritmos *inestables*, en los que pequeñas modificaciones en el conjunto de entrenamiento pueden generar clasificadores significativamente distintos. Los árboles de decisión, las redes neuronales y los algoritmos de aprendizaje basados en reglas son todos algoritmos inestables.

  Las métodos más utilizados de manipulación de los datos de entrada son *bagging* [Breiman, 1996a] y *boosting* [Freund and Schapire, 1995] (descritos en detalle en las subsecciones 2.2.1 y 2.2.2).

- **Manipulación de los atributos de entrada**. En este caso, cada clasificador se construye haciendo uso sólo de un subconjunto de variables, de tal forma que cada clasificador se especializa en un subespacio de atributos. Esta técnica debe ser utilizada cuidadosamente ya que la eliminación de atributos puede afectar de manera muy significativa al rendimiento de los clasificadores base. Un ejemplo de este tipo de técnicas es *random subspace* [Ho, 1998], en el que cada clasificador base utiliza un subconjunto de atributos seleccionado aleatoriamente de entre todas las variables.

- **Manipulación de las etiquetas de clase.** La mayoría de las técnicas de manipulación de etiquetas están enfocadas a solucionar problemas multiclase reduciéndolos a varios problemas de dos clases. Sin embargo, también existen aproximaciones para problemas de dos clases que añaden ruido a los datos mediante el intercambio aleatorio de etiquetas al construir cada clasificador [Breiman, 2000; Martínez-Muñoz and Suárez, 2005].

  Una de estas aproximaciones [Breiman, 2000] consiste en alterar las etiquetas de clase para algunos de los patrones de entrenamiento (***output flipping***), manteniendo la proporción original de cada clase dentro del conjunto. Para ello, se define el procedimiento general a partir del cual, fijada la proporción de datos cuya etiqueta será cambiada (*flip rate*), se obtiene la proporción de elementos de cada clase cuya etiqueta debe ser sustituida por la de otra cierta clase. En el caso de dos clases, tenemos:

$$p_{y_1 \leftarrow y_2} = c \cdot p(y_1) \ , \qquad p_{y_2 \leftarrow y_1} = c \cdot p(y_2) \ , \tag{2.16}$$

  donde $p_{y_1 \leftarrow y_2}$ es la probabilidad de que un ejemplo de clase y_2 pase a ser de clase y_1 y $p(y_1)$ es la proporción de elementos de clase y_1 dentro del conjunto (de la misma forma se definen los elementos de la segunda igualdad).

  De esta forma, fijado el *flip rate* deseado, fr, puede obtenerse el parámetro c (a partir de la ec. 2.16) para obtener la probabilidad de cambiar de una clase a otra

para cualquier par de clases:

$$\begin{aligned} fr &= p_{y_1 \leftarrow y_2} \cdot p(y_2) + p_{y_2 \leftarrow y_1} \cdot p(y_1) \\ &= c \cdot p(y_1) \cdot p(y_2) + c \cdot p(y_2) \cdot p(y_1) = 2c \cdot p(y_2) \cdot p(y_1) \ . \end{aligned}$$

Por tanto,

$$c = \frac{fr}{2 \cdot p(y_2) \cdot p(y_1)} \ . \tag{2.17}$$

Sustituyendo (2.17) en (2.16) podemos ver que para los problemas de dos clases, esta aproximación equivale a intercambiar entre ambas clases la mitad de la proporción de ejemplos indicada por *flip rate*. Aunque sus resultados son similares (en ocasiones mejores) que los obtenidos con *bagging*, su mayor inconveniente es que el valor óptimo de *flip rate* varía mucho de unos conjuntos a otros por lo que es necesario probar un amplio rango de valores hasta encontrar el adecuado al problema [Breiman, 2000].

La segunda aproximación relativa a la modificación de etiquetas de clase [Martínez-Muñoz and Suárez, 2005] no requiere conservar la proporción original de cada una de las clases dentro del conjunto de entrenamiento y está enfocada a utilizar un gran número de clasificadores (~ 1000) y una alta probabilidad de modificación de etiquetas. En este caso, todo ejemplo, independientemente de cual sea su clase, tiene una probabilidad p (***class switching***) de modificar su etiqueta. Por tanto, las probabilidades definidas en (2.16) para el método anterior, son ahora inmediatas para el caso de dos clases:

$$p_{y_1 \leftarrow y_2} = p \ , \qquad\qquad p_{y_2 \leftarrow y_1} = p \ . \tag{2.18}$$

De esta forma se consigue atenuar el desbalanceo en problemas en los que la proporción de clases es muy desequilibrada ya que el número de elementos que se intercambian desde la clase mayoritaria a la minoritaria es mucho mayor que los que lo hacen en el otro sentido.

Ambos métodos están enfocados a conseguir una de las cualidades deseables de los conjuntos de clasificadores: suponiendo que el error de cada clasificador base en el conjunto con las etiquetas modificadas es muy próximo a 0, los errores en el conjunto original son la proporción de elementos modificados aleatoriamente, los cuales no están correlacionados entre los distintos clasificadores. Teniendo en cuenta que para garantizar la convergencia, la proporción de ejemplos modificados en una cierta clase debe ser menor que la proporción de ejemplos no modificados ($p_{y_j \leftarrow y_i} < p_{y_i \leftarrow y_i}$), la diferencia entre ambas técnicas radica en que con la segunda aproximación el valor de $p_{y_j \leftarrow y_i}$ puede ser mayor que en el primer caso si las clases están desbalanceadas. Mientras que con el primer método el valor del *flip rate* debe ser menor que la proporción de ejemplos de la clase minoritaria, en la segunda aproximación basta con que p sea menor de 0,5.

- **Introducción de aleatoriedad en el algoritmo.** El efecto que se consigue al aplicar esta técnica es que dos ejecuciones del mismo algoritmo sobre el mismo

conjunto de datos no generan el mismo clasificador base. En el caso de los árboles de decisión, la aleatoriedad suele ser introducida añadiendo incertidumbre en la elección del mejor corte para cada nodo (seleccionar aleatoriamente entre los mejores cortes posibles, seleccionar aleatoriamente un número de variables y tomar de entre ellas el mejor corte, etc.).

En muchos de estos métodos, es necesario generar una serie de números aleatorios para construir cada uno de los clasificadores base, como ocurre por ejemplo si se consideran sólo una serie de variables tomadas aleatoriamente al seleccionar el corte en un nodo del árbol. De esta forma, el clasificador base *k-ésimo* es generado a partir de un vector de números aleatorios $\mathbf{\Theta_k}$, independiente pero con la misma distribución que los $\mathbf{\Theta_1}, \ldots, \mathbf{\Theta_{k-1}}$ vectores aleatorios utilizados para generar los $k-1$ clasificadores anteriores. Los conjuntos de clasificadores generados mediante este procedimiento y cuyo clasificador base es un árbol de decisión que contribuye en la misma medida que el resto a la decisión final del conjunto son conocidos como ***random forests*** [Breiman, 2001] (descritos en la subsección 2.2.3).

Una vez que se ha construido el conjunto de clasificadores mediante uno de los métodos anteriores, es necesario combinar sus salidas para obtener el resultado del conjunto. Aunque existen otras técnicas para **combinar las salidas de los clasificadores base**, en todos los métodos descritos en esta memoria el resultado global se obtiene ejecutando todos los clasificadores base y combinando sus salidas mediante votación ponderada (*boosting*) o no ponderada (*random forests* y *bagging*). De esta forma, la salida C de un conjunto de T clasificadores $c_1, c_2, \ldots, c_T$ cuyos pesos ω_i vienen dados por el vector columna $\boldsymbol{\omega}$ y donde la salida del clasificador i para un cierto patrón $\mathbf{x}$ viene dada por $c_i(\mathbf{x}) \in \{+1, -1\}$, se puede escribir como:

$$C(\mathbf{x}) = \text{sign}(\boldsymbol{\omega}' \cdot \mathbf{c}) \ .$$

Es importante tener en cuenta que en el caso no ponderado $\omega_i = 1/T \ \forall i$ y que cuando el producto $\boldsymbol{\omega}' \cdot \mathbf{c}$ sea 0, la clase de $\mathbf{x}$ será asignada arbitrariamente a cualquiera de las dos.

Todos los métodos basados en conjuntos de clasificadores están orientados a conseguir que la precisión del conjunto mejore la precisión de los clasificadores individuales, lo cual hace que este tipo de técnicas obtengan muy buenos resultados. Esto es debido fundamentalmente a que los conjuntos de clasificadores **salvan tres restricciones** encontradas en la construcción de modelos individuales [Dietterich, 2000a]:

- **Estadísticas.** Los algoritmos de aprendizaje son procedimientos que nos permiten encontrar funciones $f : \mathbb{R}^d \to \{1, 2, \ldots, C\}$ dentro del espacio de hipótesis $\mathcal{F}$ que se ajustan lo más posible a los datos de entrenamiento tratando de aproximar la solución real h. Sin embargo, puede darse la situación de que los datos de entrenamiento disponibles no sean suficientes y varias funciones $\hat{f}_i$ obtengan la máxima precisión para los datos disponibles. Si sólo se selecciona una de ellas,

existe el riesgo de tomar la que suponga el error de generalización más alto, mientras que si se construye un conjunto, la media entre sus salidas compensará los errores de cada una obteniendo una buena aproximación de h (figura 2.5, arriba a la izquierda).

En el caso de los árboles, la función óptima h puede venir dada por un árbol de decisión muy complejo, para el que se necesitan más datos de entrenamiento que los disponibles, por lo que construir árboles más pequeños y combinarlos constituirá una mejor aproximación que construir un solo árbol a partir de un número de datos insuficientes.

- **Computacionales.** La mayoría de los algoritmos de aprendizaje constituyen un problema de optimización que en muchos casos es resuelto realizando búsquedas locales, las cuales pueden no desembocar en el óptimo global (figura 2.5, arriba a la derecha).

  Este es el caso de los árboles de decisión, en los que encontrar la hipótesis óptima h es computacionalmente impracticable (problema NP-completo). Los métodos de construcción de árboles (como CART) toman una serie de decisiones *locales* (no tienen en cuenta las decisiones posteriores) pudiendo cometer errores que afectarán al resto del árbol a partir del punto donde se tome la decisión incorrecta. Por tanto, debido a que el problema computacional es intrínseco a los árboles de decisión, la combinación de varios árboles $\hat{f}_i$ construidos a partir de diferentes puntos de partida, constituye una mejor aproximación a h que un solo árbol construido a partir de todos los datos en el que se han podido tomar varias decisiones incorrectas.

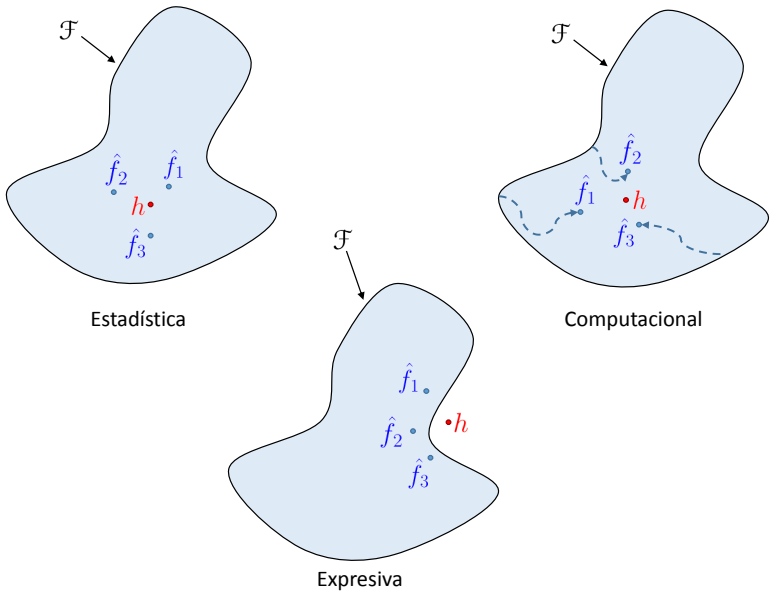

Figura 2.5: Tres razones para la eficiencia de los conjuntos de clasificadores (adaptado de [Dietterich, 2000a])

- **Expresivas.** En ocasiones, la hipótesis óptima h no es representable por ninguna de las posibles hipótesis en $\mathcal{F}$. A partir de la combinación de varias hipótesis posibles $\hat{f}_i$, el espacio de funciones $\mathcal{F}$ puede expandirse y aproximarse a la función óptima h (figura 2.5, abajo).

  En el caso de los árboles de decisión, el problema expresivo sucede debido al uso de particiones rectangulares del espacio de características ya que las decisiones en cada nodo son tomadas a partir de una sola variable. Si la frontera óptima no es ortogonal a los ejes de coordenadas, un árbol de decisión finito nunca podrá representarla. En este caso, la combinación de distintos árboles más pequeños es equivalente a un árbol mucho más grande, el cual puede constituir una buena aproximación a una frontera diagonal.

Otra forma de justificar el buen funcionamiento de los conjuntos de clasificadores es en términos del **sesgo** y la **varianza** [Breiman, 1996b]. A partir de ellos, se puede definir el error de un clasificador C como

$$PE(C) = PE(C^*) + Sesgo(C) + Varianza(C) \ , \tag{2.19}$$

donde C^* es el clasificador de Bayes y PE (*prediction error*) para el caso de clasificación se define como

$$PE(C) = \mathbb{E}_{\mathcal{T}} \left[\mathbb{E}_{\mathbf{X},Y} \left[I(C(\mathbf{X}, \mathcal{T}) \neq Y) \right] \right] \ , \tag{2.20}$$

donde I es la función indicador que devuelve un 1 si el argumento es verdadero y un 0 en caso contrario y $\mathbf{X}, Y$ es la distribución de los datos de entrenamiento. De esta forma, $\mathbb{E}_{\mathbf{X},Y}$ es el valor esperado sobre dicha distribución, $\mathcal{T}$ es cualquier conjunto de entrenamiento construido a partir de ella y $\mathbb{E}_{\mathcal{T}}$ es la esperanza sobre todos los posibles conjuntos de entrenamiento.

El **Error de Bayes** $PE(C^*)$ es el error intrínseco al problema de clasificación como consecuencia de aquellas regiones del espacio de características en las que coexisten patrones de dos o más clases distintas. No puede reducirse y constituye, por tanto, la cota mínima del error. Este error mínimo, que es difícilmente medible en conjuntos de datos reales (no se conoce la distribución de las clases), viene dado por el clasificador de Bayes C^*, el cual clasifica cada patrón $\mathbf{x}$ como la clase más probable para ese vector:

$$C^*(\mathbf{x}) = \arg\max_y p(y|\mathbf{x}) \ . \tag{2.21}$$

Por tanto, con el objetivo de disminuir el error de clasificación, el sesgo y la varianza son los únicos términos que pueden reducirse. Ambos tienen su origen en problemas de regresión, por lo que antes de pasar a su definición para clasificación, se describen ambos conceptos enfocados a regresión:

- **Sesgo.** El sesgo de un cierto algoritmo se obtiene a partir de la diferencia entre el valor real de cada patrón y la media de las predicciones obtenidas para ese patrón usando distintos conjuntos de entrenamiento. Mide, por tanto, el error en la tendencia central del algoritmo.

- **Varianza.** La varianza se obtiene a partir de la diferencia entre la predicción media obtenida para un patrón usando distintos conjuntos de entrenamiento y la predicción obtenida en cada uno de ellos. Es decir, mide las variaciones en las predicciones cuando se usan distintos conjuntos de entrenamiento.

De esta forma, familias de algoritmos predominantemente lineales tienen un sesgo alto (las predicciones no son muy precisas) pero en cambio su varianza es baja (las variaciones entre las predicciones obtenidas cuando se usan distintos conjuntos de entrenamiento no son muy cambiantes ya que el número de parámetros a ajustar es pequeño). En cambio, al añadir no linealidad a los algoritmos, el sesgo disminuye pero la varianza aumenta (los modelos son más precisos pero también más ajustados al conjunto de entrenamiento).

En el caso de clasificación, la idea de sesgo y varianza es la misma que en el caso de regresión pero se necesita adaptar el concepto de *predicción media* de un patrón sobre distintos conjuntos de entrenamiento. Para ello, se definen [Breiman, 1996b]:

$$Q(y|\mathbf{x}) = \mathbb{E}_{\mathcal{T}}\left[I(C(\mathbf{x},\mathcal{T}) = y)\right] \ , \tag{2.22}$$

$$C_A(\mathbf{x}) = \arg\max_y Q(y|\mathbf{x}) \ , \tag{2.23}$$

donde $Q(y|\mathbf{x})$ corresponde a la proporción de los posibles conjuntos de entrenamiento $\mathcal{T}$ que clasifican a $\mathbf{x}$ como y, $C(\mathbf{x},\mathcal{T})$ es la predicción de C sobre $\mathbf{x}$ cuando C ha sido entrenado con $\mathcal{T}$ y C_A es la clase más votada entre los distintos $\mathcal{T}$. C_A se utiliza para definir un clasificador no sesgado:

$$C \text{ es no sesgado en } \mathbf{x} \Leftrightarrow C_A(\mathbf{x}) = C^*(\mathbf{x}) \ . \tag{2.24}$$

Por tanto, $C(\mathbf{x},\mathcal{T})$ es no sesgado en $\mathbf{x}$ cuando, al aplicarse sobre todos los $\mathcal{T}$ posibles, $C(\mathbf{x},\mathcal{T})$ selecciona la clase más probable más veces que cualquier otra clase. De esta forma, se puede definir U como el conjunto de todos los $\mathbf{x}$ para los que C es no sesgado (conjunto no sesgado) y B como su complementario (conjunto sesgado).

A partir de los conjuntos anteriores, se define el sesgo y la varianza de C como:

$$Sesgo(C) = \mathbb{E}_{\mathbf{X},Y}\left[I(C^*(\mathbf{X}) = Y), \mathbf{X} \in B\right] - \mathbb{E}_{\mathcal{T}}\left[\mathbb{E}_{\mathbf{X},Y}\left[I(C(\mathbf{X},\mathcal{T}) = Y), \mathbf{X} \in B\right]\right] \ , \tag{2.25}$$

$$Varianza(C) = \mathbb{E}_{\mathbf{X},Y}\left[I(C^*(\mathbf{X}) = Y), \mathbf{X} \in U\right] - \mathbb{E}_{\mathcal{T}}\left[\mathbb{E}_{\mathbf{X},Y}\left[I(C(\mathbf{X},\mathcal{T}) = Y), \mathbf{X} \in U\right]\right] \ . \tag{2.26}$$

De esta forma, el sesgo se calcula sobre el conjunto de puntos sobre los que el clasificador es sesgado (B). A partir de estos puntos, el sesgo es la diferencia entre aquellos que podían haber sido etiquetados correctamente (ya que el clasificador de Bayes lo hace) y la proporción media de estos que son etiquetados correctamente usando distintos conjuntos de entrenamiento. La varianza, en cambio, se calcula a partir del conjunto no sesgado (U). Mide la diferencia entre la proporción de aciertos del clasificador de Bayes (que en este caso coincide con el clasificador agregado C_A) y el acierto medio del

clasificador para cada uno de los conjuntos de entrenamiento; de esta forma, cuanta más variabilidad exista al usar distintos conjuntos de entrenamiento, mayor será esta diferencia.

A partir de esta definición, si $C(\mathbf{x}, \mathcal{T})$ y $C^*(\mathbf{x})$ son sustituidos por $C_A(\mathbf{x})$ en la expresión (2.26), la varianza puede reducirse hasta 0 ya que ahora $C_A(\mathbf{x})$ no depende de $\mathcal{T}$ y por tanto, en el segundo término, su valor medio es justamente $C_A(\mathbf{x})$. Por tanto, se puede concluir a partir de estas definiciones que clasificadores que combinan por votación las decisiones de varios clasificadores individuales tenderán a reducir el error de varianza.

Además, los clasificadores inestables, como son los árboles de decisión, se caracterizan por su varianza alta ya que pequeñas modificaciones en el conjunto de entrenamiento pueden generar árboles con estructuras muy distintas. De esta forma, los conjuntos formados por árboles de decisión tienen un amplio margen de mejora con respecto a la varianza. Sin embargo, en cuanto al sesgo, no hay garantía de que sea menor en el conjunto que en cada uno de los clasificadores individuales. Por el contrario, clasificadores estables, como pueden ser los discriminantes lineales o vecinos próximos, no se ven muy afectados por cambios en el conjunto de entrenamiento por lo que no existen grandes diferencias entre $C_A(\mathbf{x})$ y $C(\mathbf{x}, \mathcal{T})$ y, por tanto, ni la varianza original es alta ni su disminución significativa.

Una vez expuestas las condiciones deseables para los conjuntos de clasificadores, las técnicas para construirlos, cómo deben ser combinadas sus salidas y el por qué de las mejoras obtenidas a partir de ellos, se detallan a continuación las características de los tres algoritmos principales para la construcción de conjuntos de clasificadores: *bagging* (subsección 2.2.1), *boosting* (subsección 2.2.2) y *random forests* (subsección 2.2.3).

2.2.1. Bagging

Uno de los métodos más extendidos y a la vez más sencillos para construir conjuntos de clasificadores es *bagging* [Breiman, 1996a]. El término *bagging* es un acrónimo de *bootstrap aggregating* ya que el algoritmo construye cada uno de los clasificadores del conjunto a partir de lo que se conoce como una muestra ***bootstrap***, las cuales se generan tomando del conjunto de entrenamiento tantos elementos con reemplazamiento como éste contenga. De esta forma, se generan T conjuntos del mismo tamaño que el conjunto original (uno por cada clasificador), en los que algunos ejemplos no aparecen mientras que otros están presentes más de una vez. Una vez construido el conjunto, las salidas de cada uno de los clasificadores se combinan por votación no ponderada.

Debido a que cada clasificador individual se construye a partir de una muestra del conjunto de entrenamiento en la que se toman N elementos con reemplazamiento sobre el conjunto total de N elementos, no todos son usados en cada clasificador, lo cual está enfocado a conseguir independencia entre los clasificadores. De esta forma, la precisión de cada uno de los clasificadores individuales es menor que la que se obtendría si se construyera un clasificador con todo el conjunto pero, al combinarlos, se compensan

los errores de cada uno y se mejora la precisión que obtendríamos con uno solo que utilizara todos los datos. La proporción de elementos distintos que hay en cada muestra *bootstrap* es igual a 1 menos la probabilidad de que un elemento no aparezca:

$$P = 1 - \left(1 - \frac{1}{N}\right)^N \simeq 1 - e^{-1} \simeq 0,63 \text{ para } N \text{ grande } (N \gg 1) \ , \tag{2.27}$$

donde $(1 - 1/N)^N$ es la probabilidad de que un elemento no aparezca en N extracciones sobre N datos equiprobables, la cual es aproximadamente e^{-1} si N es suficientemente grande. Por tanto, cada clasificador se construye con una muestra en la que el 63,2 % de los datos son distintos.

A partir de los datos que no se usan al construir cada clasificador (aproximadamente un 37 % del total), se puede obtener una estimación del error de generalización, la cual se conoce como estimación ***out-of-bag*** [Breiman, 2001]. De la misma forma que cada clasificador no utiliza en torno a un 37 % de los ejemplos, cada ejemplo del entrenamiento no se ha utilizado para construir en torno a un 37 % de los clasificadores. De esta forma, cada uno de los ejemplos se puede clasificar utilizando sólo dichos clasificadores, los cuales no han visto el ejemplo antes, obteniendo así una estimación válida de su error. Promediando el error para todos los ejemplos del conjunto de entrenamiento se obtiene el error de generalización del conjunto sin necesidad de un conjunto de test adicional ya que la estimación *out-of-bag* es tan precisa como un conjunto de test del mismo tamaño que el conjunto de entrenamiento. La única restricción es que la estimación obtenida se basa únicamente en aproximadamente un tercio de los clasificadores y, debido a que el error disminuye al aumentar el número de clasificadores, la estimación *out-of-bag* tiende a sobrestimar el error de generalización. Para conseguir una estimación no sesgada sería necesario seguir construyendo clasificadores más allá del punto donde el error de entrenamiento converge.

Bagging constituye, dentro de las técnicas anteriormente descritas, una aproximación basada en la manipulación de los datos de entrenamiento, en las cuales es vital que el clasificador base utilizado sea inestable (varianza alta). De hecho, *bagging* reduce muy significativamente el error de generalización para este tipo de clasificadores mientras que puede no afectar (o incluso empeorar) el rendimiento de clasificadores base estables. Esto sucede así debido a que *bagging* reduce la varianza del algoritmo base, estabilizando la clasificación obtenida mediante votación y consiguiendo que las predicciones no fluctúen mucho al usar distintos conjuntos de entrenamiento. Por tanto, los árboles de decisión son buenos candidatos sobre los que aplicar *bagging* aunque esta técnica haga perder una de sus grandes ventajas, la intuitiva interpretación de sus resultados en forma de reglas.

Otra ventaja de este método es su alta escalabilidad: cada uno de los clasificadores puede ser construido (y evaluado) en paralelo ya que la construcción de cada uno de ellos es completamente independiente de la de los demás, lo cual implica que el tiempo necesario para construir un conjunto de clasificadores no aumenta proporcionalmente al número de clasificadores utilizados.

2.2.2. Boosting

Otra de las técnicas más utilizadas y eficaces para construir conjuntos de clasificadores es *boosting* [Freund and Schapire, 1995; Freund and Shapire, 1996]. *Boosting* es un algoritmo adaptativo en el que cada clasificador se construye en base a los resultados obtenidos en los clasificadores previos mediante la asignación de pesos a cada uno de los ejemplos de entrenamiento: patrones que han sido incorrectamente clasificados por los clasificadores anteriores tendrán más importancia a la hora de construir el nuevo clasificador. De esta forma, los clasificadores se centran en aquellos ejemplos que son más difíciles de etiquetar correctamente por el conjunto. En este caso, una vez construidos los clasificadores, las salidas se combinan de forma ponderada según la importancia de cada clasificador individual para obtener el resultado del conjunto.

Algoritmo 1: Algoritmo *AdaBoost.M1*

Entradas *Conjunto* $\mathcal{L} = \{(\mathbf{x_n}, y_n) \mid n = 1, 2, \ldots, N, \mathbf{x_n} \in \mathbb{R}^d, y_n \in \{1, 2\}\}$
Clasificador débil ***WeakLearn***
Número de clasificadores T

for $n := 1$ **to** N **do**
  $\omega_{1,n} := \frac{1}{N}$

for $t := 1$ **to** T **do**
  $c_t := \mathbf{WeakLearn}(\mathcal{L}, \boldsymbol{\omega_t})$
  $\epsilon_t := \sum_{n=1}^{N} \omega_{t,n} \cdot I(c_t(\mathbf{x_n}) \neq y_n)$
  if $\epsilon_t \geq 0{,}5$ **then**
    $T := t - 1$
    abort loop
  $\beta_t := \frac{\epsilon_t}{1-\epsilon_t}$
  for $n := 1$ **to** N **do**
    if $c_t(\mathbf{x_n}) = y_n$ **then**
      $\tilde{\omega}_{t+1,n} := \beta_t \cdot \omega_{t,n}$
    else
      $\tilde{\omega}_{t+1,n} := \omega_{t,n}$
  $Z_t := \sum_{n=1}^{N} \tilde{\omega}_{t+1,n}$
  for $n := 1$ **to** N **do**
    $\omega_{t+1,n} := \frac{\tilde{\omega}_{t+1,n}}{Z_t}$

Salida: $C(\mathbf{x}) = \arg\max_y \sum_{t=1}^{T} log(1/\beta_t) I(c_t(\mathbf{x}) = y)$

En su origen, *boosting* se definió como una técnica mediante la cual se reduce significativamente el error de cualquier algoritmo débil (entendiendo como algoritmo débil todo aquel cuyo error es un poco menor que el clasificador aleatorio) pero, en la práctica, es un método que se combina con árboles de decisión, los cuales no son

considerados débiles. Según cuál sea el algoritmo utilizado, existen dos variantes de *boosting*: *reweighting* y *resampling*. En el caso de que el algoritmo se pueda adaptar para manejar datos ponderados, todos los ejemplos y sus respectivos pesos son utilizados por cada clasificador base, el cual es el encargado de tenerlos en cuenta (*boosting by reweighting*). En caso de que el algoritmo no pueda gestionar datos ponderados, es necesario realizar un muestreo con reemplazamiento en función de los pesos de cada ejemplo de tal forma que el clasificador recibe un conjunto de datos sin pesar (*boosting by resampling*).

Aunque existen numerosas variantes basadas en la idea anteriormente expuesta, el algoritmo más utilizado y popular, por su simplicidad de implementación y porque no requiere conocimiento previo sobre la precisión de los clasificadores base, es ***AdaBoost*** (*adaptive boosting*), del que existen dos versiones distintas: *AdaBoost.M1* y *AdaBoost.M2*. Ambas son equivalentes para problemas de dos clases por lo que sólo se describirá la versión más simple, *AdaBoost.M1*. Sin pérdida de generalidad, se supone que en este caso el algoritmo es utilizado para *boosting by reweighting*.

El algoritmo *AdaBoost.M1* (algoritmo 1) parte de un conjunto de datos $\mathcal{L}$, un algoritmo de aprendizaje (denotado como *WeakLearn*) y el número de clasificadores a construir T. En cada iteración se construye un clasificador t para el que cada ejemplo $\mathbf{x_n}$ tiene un determinado peso $\omega_{t,n}$, que en el caso del primer clasificador son inicializados equiprobables. Cada uno de los clasificadores t tiene como objetivo minimizar el error ϵ_t en base a la distribución $\boldsymbol{\omega_t}$ y es a partir de este ϵ_t (sólo si éste es menor de 0,5) a partir del cual se distribuyen los pesos para el siguiente clasificador, $\boldsymbol{\omega_{t+1}}$. Esta redistribución de pesos se realiza de tal manera que el error obtenido por t para los datos con pesos $\boldsymbol{\omega_{t+1}}$ sea exactamente 0,5; con este procedimiento se consigue que los pesos de ejemplos correctamente clasificados por clasificadores previos sean más bajos mientras que los más difíciles de clasificar tengan pesos más altos. Una vez todos los clasificadores son construidos, el clasificador final es una combinación de todos ellos en los que su peso es inversamente proporcional al error que cometieron en su construcción. De esta forma, clasificadores con menor error son más importantes en el conjunto final. Por tanto, *boosting*, dentro de las diferentes técnicas de construcción de conjuntos de clasificadores, es una técnica basada en la manipulación de los datos de entrenamiento.

En el pseudocódigo se observa que una vez que se obtiene un clasificador con error mayor de 0,5, éste es descartado y se interrumpe la ejecución del algoritmo. Esto sucede así ya que esta condición es necesaria para que el error del conjunto de clasificadores sobre el conjunto de entrenamiento cumpla lo siguiente [Freund and Schapire, 1995]:

$$\epsilon = \frac{\#\{n \mid C(\mathbf{x_n}) \neq y_n\}}{N} \leq e^{\left(-2\sum_{t=1}^{T}\gamma_t^2\right)} , \tag{2.28}$$

donde $\epsilon_t = 1/2 - \gamma_t$. De esta forma, el error del conjunto de clasificadores disminuye de forma exponencial y la mejora de cada uno de los clasificadores individuales contribuye en la disminución de la cota superior de error del conjunto. Aunque para problemas multiclase esta restricción puede resultar difícil de cumplir, para el caso de los problemas de dos clases implica ser sólo ligeramente mejor que un clasificador aleatorio, lo

cual, en caso de no cumplirse, puede resolverse intercambiando las etiquetas de ambas clases.

En cuanto a la redistribución de pesos, se puede comprobar que se hace de tal forma que el error que obtendría el clasificador t para los datos con pesos $\boldsymbol{\omega_{t+1}}$ es exactamente 0,5; es decir, el peso de los ejemplos que clasifica correctamente es el mismo que el de los ejemplos mal clasificados si los datos están distribuidos según $\boldsymbol{\omega_{t+1}}$:

$$\sum_{n|c_t(\mathbf{x_n})\neq y_n} \omega_{t+1,n} = \sum_{n|c_t(\mathbf{x_n})\neq y_n} \frac{\tilde{\omega}_{t+1,n}}{Z_t} = \frac{1}{Z_t} \sum_{n|c_t(\mathbf{x_n})\neq y_n} \omega_{t,n} = \frac{\epsilon_t}{Z_t} \,, \tag{2.29}$$

$$\sum_{n|c_t(\mathbf{x_n})= y_n} \omega_{t+1,n} = \sum_{n|c_t(\mathbf{x_n})= y_n} \frac{\tilde{\omega}_{t+1,n}}{Z_t} = \frac{1}{Z_t} \sum_{n|c_t(\mathbf{x_n})= y_n} \beta_t \cdot \omega_{t,n} = \frac{1}{Z_t} \frac{\epsilon_t}{1-\epsilon_t}(1-\epsilon_t) = \frac{\epsilon_t}{Z_t} \,. \tag{2.30}$$

La ecuación (2.29) corresponde al peso de los ejemplos mal clasificados por c_t cuando los datos están distribuidos según $\boldsymbol{\omega_{t+1}}$ mientras que en (2.30) se calcula el correspondiente a los ejemplos correctamente clasificados. Como ambos son iguales y deben sumar 1, se deduce que ambas expresiones son iguales a 0,5. Así es como *AdaBoost* consigue que los datos mal clasificados por el clasificador t, que con la distribución $\boldsymbol{\omega_t}$ tenían un error menor de 0,5, incrementen su peso dentro de la muestra hasta suponer un error del 0,5 para el clasificador t.

Por tanto, aunque *boosting* no exija explícitamente (como en el caso de *bagging*) que los clasificadores base sean inestables, si c_{t-1} fuera igual que c_t, el esquema de redistribución de pesos utilizado haría que $\epsilon_t = 0{,}5$ y, por tanto, $\beta_t = 1$, lo cual provocaría que no se actualizasen los pesos en la iteración t. Por tanto, que los clasificadores utilizados no sean *demasiado* estables es una característica deseable para *boosting*. De la misma manera, clasificadores que se ajustan completamente a los datos de entrenamiento tampoco son muy adecuados para *boosting* ya que harían que ϵ_t fuese 0, forzando al algoritmo a terminar prematuramente [Quinlan, 1996]. Por tanto, árboles de decisión con poda son muy convenientes y obtienen muy buenos resultados en conjunción con *boosting*. Otra ventaja de los árboles de decisión es que se trata de un algoritmo que se puede adaptar fácilmente para tratar ejemplos con distintos pesos y poder aplicar *boosting by reweighting*, que parece ser más efectivo que *boosting by resampling* [Quinlan, 1996].

En cuanto a los resultados obtenidos, *boosting* constituye uno de los mejores algoritmos para construir conjuntos de clasificadores [Dietterich, 2000b; Freund and Shapire, 1996; Quinlan, 1996]. Esto es debido a que *boosting* combina dos efectos: reduce el sesgo del clasificador base debido a que los fuerza a no cometer los mismos errores (corrige los errores en la tendencia central del algoritmo) y también reduce la varianza al combinar mediante voto distintas hipótesis [Freund and Shapire, 1996]. Sin embargo, presenta problemas de generalización en presencia de datos con ruido (*outliers*, errores en el valor de algunos atributos o en algunas etiquetas de clase) ya que los clasificadores se centran en los ejemplos difíciles sin tener en cuenta si son datos válidos o no. En

cambio, *bagging* (subsección 2.2.1) parece capaz de explotar el ruido de los datos para generar clasificadores más diversos obteniendo muy buenos resultados para este tipo de conjuntos de datos [Dietterich, 2000b]. Por último, otra desventaja de *boosting* es que no es paralelizable ya que cada clasificador se construye en base a los resultados obtenidos por el clasificador previo.

2.2.3. Random Forests

Como ya se ha expuesto anteriormente, existen diversas técnicas de construcción de conjuntos de clasificadores, las cuales están enfocadas a aumentar la diversidad entre los clasificadores individuales (sección 2.2). En muchas ocasiones, la forma de llevar a cabo estas técnicas es a partir de un conjunto de números aleatorios:

- En *bagging* [Breiman, 1996a] cada clasificador se construye a partir de una muestra *bootstrap*, la cual se genera utilizando tantos números aleatorios como elementos tenga el conjunto de entrenamiento.
- En *random subspace* [Ho, 1998] cada clasificador base utiliza sólo un subconjunto de atributos seleccionado aleatoriamente de entre el total de las variables. De esta forma, cada clasificador se restringe a un subespacio aleatorio de atributos.
- Las técnicas *output flipping* y *class switching* [Breiman, 2000; Martínez-Muñoz and Suárez, 2005] se basan en la manipulación aleatoria de las etiquetas de clase, por lo que necesitan de números aleatorios para seleccionar los datos cuyas etiquetas serán cambiadas.
- *Randomization* [Dietterich, 2000b] introduce aleatoriedad en el algoritmo de aprendizaje construyendo conjuntos de clasificadores con árboles de decisión en los que el valor de corte es seleccionado aleatoriamente entre los F mejores cortes posibles.

Si el clasificador base utilizado es un árbol de decisión, el concepto *random forest* [Breiman, 2001] engloba todas estas técnicas. Cualquier conjunto de clasificadores en el que el clasificador base es un árbol de decisión construido a partir de un vector de números aleatorios $\mathbf{\Theta_t}$, donde los $\{\mathbf{\Theta_t}\}$ son independientes e idénticamente distribuidos y el resultado del clasificador final se obtiene mediante votación no ponderada se conoce como *random forest*.

El objetivo de este tipo de métodos es, como en todo conjunto de clasificadores, inyectar al algoritmo la aleatoriedad justa para maximizar la independencia de los árboles manteniendo una precisión razonable. En el caso de los *random forests*, estas cualidades se miden a nivel del conjunto y se denotan como **fuerza** y **correlación**. La fuerza del conjunto de clasificadores se define como

$$s = \mathbb{E}_{\mathbf{X},Y}\ mr(\mathbf{X}, Y)\ , \tag{2.31}$$

donde $mr(\mathbf{X}, Y)$ es la función de margen de un *random forest* que, en el caso de dos clases, se define como

$$mr(\mathbf{X}, Y) = \mathbb{E}_\Theta \left[c(\mathbf{X}, \Theta) = Y\right] - \mathbb{E}_\Theta \left[c(\mathbf{X}, \Theta) \neq Y\right] = 2 \cdot \mathbb{E}_\Theta \left[c(\mathbf{X}, \Theta) = Y\right] - 1 \ , \quad (2.32)$$

donde $\mathbb{E}_\Theta \left[c(\mathbf{x}, \Theta) = y\right]$ es el límite de la proporción de árboles c_t que, dado un patrón $\mathbf{x}$, lo clasifican correctamente (al aumentar T):

$$\frac{1}{T} \sum_{t=1}^{T} I(c_t(\mathbf{x}, \Theta_t) = y) \rightarrow \mathbb{E}_\Theta \left[c(\mathbf{x}, \Theta) = y\right] \ . \quad (2.33)$$

El concepto de margen en los conjuntos de clasificadores sirve para medir la *seguridad* con la que el conjunto acierta o se equivoca en su predicción ya que es la diferencia entre la proporción de árboles que aciertan y los que se equivocan. De esta forma, el margen es un valor definido en el intervalo $[-1, +1]$ siendo positivo cuando el ejemplo se ha clasificado correctamente y negativo en caso contrario. Por tanto, cuanto mayor sea el margen medio obtenido por un *random forest* sobre un conjunto de datos, mayor será su fuerza.

Por otro lado, también en el caso de problemas de dos clases, la correlación entre los árboles dentro del conjunto se mide de la siguiente forma:

$$\bar{\rho} = \mathbb{E}_{\Theta,\Theta'} \left[\rho(c(\cdot, \Theta), c(\cdot, \Theta'))\right] \ , \quad (2.34)$$

donde ρ corresponde al coeficiente de correlación entre dos variables aleatorias, donde las etiquetas de clase deberán fijarse como $+1$ y -1 y donde Θ y Θ' son independientes con la misma distribución. En este caso, es deseable que la correlación del conjunto sea mínima para aumentar así la independencia entre los distintos árboles del conjunto. La estimación de estos parámetros de fuerza y correlación a partir de datos experimentales se describe en la subsección 4.3.2.

Aunque todas las técnicas anteriormente expuestas están enfocadas a disminuir la correlación de los clasificadores base manteniendo su precisión y todas ellas consiguen disminuir significativamente (unas en mayor medida que otras) el error del clasificador base, ninguna de ellas consigue, en general, el rendimiento obtenido por *AdaBoost* (subsección 2.2.2) o por cualquier otro método que asigne pesos de forma adaptativa durante la construcción del conjunto de clasificadores. Sin embargo, los *random forests* propuestos en [Breiman, 2001] sí logran igualar (y, en ocasiones, mejorar) la precisión conseguida por *AdaBoost*. Estos nuevos métodos, de resultados muy similares, se denominan *Forest-RI* y *Forest-RC*. Ambos métodos se basan en la combinación de *bagging* y árboles de decisión no podados en los que las decisiones en cada nodo se toman considerando sólo un subconjunto de variables seleccionadas aleatoriamente:

- ***Forest-RI.*** El pseudocódigo 2 describe este procedimiento, en el que la función *MuestreoBootstrap* devuelve una muestra en la que se han tomado N elementos con reemplazamiento del conjunto total de N elementos y *ConstruyeArbolAleatorio* devuelve un árbol en el que, en cada nodo, F variables son seleccionadas aleatoriamente y el corte elegido corresponde al mejor entre todos los posibles para el subconjunto de variables seleccionado.

- ***Forest-RC.*** En este caso es necesario definir dos parámetros: L, el número de variables que se combinarán en cada corte, y F, el número de combinaciones que se generarán para cada nodo. De esta forma, cada árbol, construido a partir de una muestra *bootstrap*, selecciona aleatoriamente L variables en cada nodo y genera a partir de ellas F combinaciones lineales cuyos coeficientes son seleccionados a partir de una distribución aleatoria uniforme en el intervalo $[-1, +1]$. El corte seleccionado, que en este caso será oblicuo (si $L > 1$), será el mejor entre las F combinaciones lineales creadas.

  En el caso de problemas con pocas variables, si se utiliza *Forest-RI* y el valor de F elegido corresponde a una proporción importante de ellas, puede suceder que la correlación entre los árboles sea muy alta. Por ello, para este tipo de problemas en los que los datos tienen pocos atributos, es más recomendable el uso de *Forest-RC* ya que de esta forma nuevas variables son creadas a partir de combinaciones lineales aleatorias de las ya existentes.

Algoritmo 2: Algoritmo *Forest-RI*

Entradas *Conjunto* $\mathcal{L} = \{(\mathbf{x_n}, y_n) \mid n = 1, 2, \ldots, N, \mathbf{x_n} \in \mathbb{R}^d, y_n \in \{1, 2\}\}$
Número de árboles T
Número de variables a seleccionar en cada nodo F

for $t := 1$ **to** T **do**
  $L_{bt} := MuestreoBootstrap(\mathcal{L})$
  $c_t := ConstruyeArbolAleatorio(L_{bt}, F)$

Salida: $C(\mathbf{x}) = \arg\max_y \sum_{t=1}^{T} I(c_t(\mathbf{x}) = y)$

De esta forma (restringiéndonos al caso de *Forest-RI*) los dos únicos parámetros que se deben ajustar para desarrollar estos algoritmos son el número de variables a seleccionar en cada nodo, F, y el número de árboles a construir, T:

- Uno de los resultados más sorprendentes de los experimentos llevados a cabo en [Breiman, 2001] es que el error de generalización no es muy sensible al número de variables seleccionadas aleatoriamente en cada nodo. Las medidas de fuerza y correlación del conjunto de clasificadores justifican este hecho, ya que si se mide la fuerza y la correlación de un conjunto de clasificadores construido con distintos valores de F, se observa que la fuerza aumenta hasta un determinado punto (significativamente menor que el número total de variables) a partir del cual se estabiliza, mientras que la correlación entre los clasificadores siempre aumenta al aumentar el valor de F. Por tanto, existe un cierto valor de F óptimo (dependiente del problema) en el que para la fuerza máxima, la correlación de los clasificadores es mínima aunque en cualquier caso las oscilaciones en el error al utilizar distintos valores de F no son muy significativas.

- Al aumentar el número de árboles, el error de generalización converge, en el caso

de dos clases, a

$$\mathbb{E}_{\mathbf{X},Y}[I(\mathbb{E}_{\Theta}[c(\mathbf{X},\Theta)=Y]-\mathbb{E}_{\Theta}[c(\mathbf{X},\Theta)\neq Y]<0)]\,. \tag{2.35}$$

Es decir, la proporción de datos en la que el *random forest* se equivoca converge según aumenta el número de árboles. Por tanto, a partir de un cierto número, el error de generalización se estabiliza de tal forma que aumentar el tamaño del conjunto nunca provocará un sobreajuste del modelo a los datos.

Por tanto, en resumen, las seis grandes ventajas de los nuevos métodos propuestos son las siguientes:

1. La precisión obtenida mediante estos métodos es igual o mejor que la conseguida por *AdaBoost*. Este hecho parece indicar que este tipo de procedimientos puede suponer también, al igual que *AdaBoost*, una reducción del sesgo en el clasificador.

2. Es robusto frente a datos con ruido y a la presencia de *outliers*. Al tratarse de un método basado en *bagging*, hereda de éste su estabilidad frente al ruido a diferencia de *AdaBoost*, que es muy sensible a estos problemas.

3. Es más rápido computacionalmente que *bagging*, que evalúa todos los posibles cortes de todas las variables, y que *boosting*, que es secuencial.

4. Al utilizar *bagging* permite obtener una estimación del error de generalización sin necesidad de un conjunto de test adicional, a partir de los ejemplos *out-of-bag* (sección 2.2.1).

5. Es sencillo de implementar y paralelizable.

6. Nunca se sobreajusta al conjunto de entrenamiento.

Capítulo 3

Clasificación de fraude en medios de pago

El problema de clasificación de fraude constituye un ejemplo muy particular dentro del aprendizaje supervisado. En este capítulo se realiza una descripción de sus principales características y peculiaridades así como de las medidas de rendimiento comúnmente utilizadas para su evaluación.

3.1. Características del problema

La detección de fraude en medios de pago constituye un problema de clasificación cuyo objetivo es la identificación de transacciones fraudulentas basándose únicamente en la información contenida en ellas y partiendo de la base de que los patrones de comportamiento de un defraudador son significativamente distintos a los del cliente real. El fraude ocurre cuando no es el cliente legítimo quien realiza la operación sino un tercero que ha conseguido operar como si fuese el verdadero cliente, habiendo salvado ya todos los mecanismos de seguridad del banco. Por tanto, la detección de fraude como problema de clasificación se trata de la última línea de defensa de las entidades financieras. Dentro del problema del fraude en medios de pago es importante distinguir dos tipos:

1. **Operaciones con tarjeta.** Las transacciones son las operaciones realizadas con tarjetas de crédito o débito, las cuales son utilizadas para comprar bienes y servicios, tanto en establecimientos físicos cono en Internet. El fraude en este tipo de operaciones ocurre generalmente cuando las tarjetas son copiadas, ya que en caso de robo, el cliente suele advertir la pérdida antes de que el defraudador pueda actuar. La forma más habitual de clonar tarjetas es instalar dispositivos en los terminales de las tiendas o en los cajeros automáticos, los cuales guardan la información de la banda magnética al realizar una operación normal. En el caso de compras por Internet, el fraude es incluso más fácil ya que con conocer los datos de la tarjeta es suficiente sin necesidad de que ésta esté presente.

2. **Transferencias entre cuentas.** Las transacciones son transferencias entre cuentas que se realizan a través de banca por Internet, teléfono o en las propias sucursales. En este caso, el defraudador consigue acceder a las cuentas del cliente mediante sus credenciales, las cuales ha podido obtener de diferentes maneras. Una técnica muy extendida es el *phising*, que consiste en el envío de un correo electrónico en el que se facilita un enlace a un portal falso pero idéntico al del banco del usuario donde se solicitan al cliente las claves de acceso a sus cuentas, que son en realidad enviadas al defraudador.

Será sobre este último tipo de transacciones sobre las que se realizará la parte principal de este estudio ya que constituye un problema que difiere en algunos aspectos del problema típico de clasificación de fraude (operaciones con tarjeta). Una de las principales características del problema del fraude es el **desbalanceo de clases**, ya que la proporción de ejemplos de fraude respecto de las transacciones genuinas es mínima, pero es aún más pequeña en el caso de transferencias entre cuentas. En el caso de tarjetas, aproximadamente 1 de cada 3000 ejemplos es fraude mientras que para el caso de transferencias es fraude 1 de cada 10 000.

Otra de las principales peculiaridades del problema es el **volumen de los datos** en cuanto al número de ejemplos disponibles. En una entidad financiera media, el número de operaciones con tarjeta al día es del orden de varios millones y el número de transferencias de varios cientos de miles, por lo que la generación de modelos que tengan en cuenta varios meses de transacciones implica el procesamiento de millones de patrones de una forma eficiente para poder manejar tiempos razonables al construir y validar los modelos. Además, debido al gran desbalanceo de clases, es necesario construir modelos con un gran número de ejemplos para que la muestra de los casos de fraude sea representativa.

Por otro lado, un volumen tan grande de datos permite usar un **gran número de variables** para definir los vectores de características y a la vez disponer de suficientes ejemplos para cubrir el espacio definido por ellas, evitando el problema de la *maldición de la dimensionalidad* (el aumento en la dimensión de los datos hace necesarios muchos más datos para disponer de ejemplos que cubran todo el espacio de características).

En cuanto al origen de los datos utilizados en este estudio, es importante remarcar que se trata de **transacciones reales** realizadas en distintos bancos de distintos lugares del mundo. Por tanto, se trata de información altamente sensible y confidencial de la que no se puede ofrecer ninguna descripción. Además, al tratarse de transacciones reales, son datos con mucho **ruido**, debido principalmente a dos factores:

- El fraude puede no ser marcado como consecuencia de cualquier error en el procedimiento de reporte del fraude por parte del cliente al banco, lo cual supone transacciones de fraude etiquetadas como genuinas.
- Hay transacciones genuinas que presentan las mismas características que una transacción fraudulenta. En ocasiones, el cliente legítimo puede realizar compras o transferencias anómalas (muy diferentes a lo habitual) cuyas características

sean más parecidas a una transacción de fraude que a una transacción genuina. Esto se traduce en patrones genuinos dentro de regiones del espacio en las que las transacciones son en su mayoría fraude.

Se trata además de un problema **cambiante con el paso del tiempo**. Los defraudadores están constantemente modificando su forma de actuar y explorando nuevas formas de cometer fraude, por lo que los modelos diseñados hoy pueden no resultar útiles pasado un cierto tiempo. Esta situación es más extrema cuando el sistema de detección de fraude funciona en línea ya que el defraudador detecta qué comportamientos son alertados por el modelo y cambia su conducta hacia nuevas prácticas que el sistema no pudo aprender cuando se entrenó. Por tanto, los modelos construidos con datos hasta un cierto punto en el tiempo deben ser eficaces al clasificar datos que aparecen a partir de ese instante, los cuales pueden tener características distintas a las que tenían los datos de entrenamiento. De esta forma, la suposición general en los problemas de clasificación de que los datos de test tienen la misma distribución que los de entrenamiento, puede no cumplirse en el caso de la clasificación de fraude.

3.2. Medidas de rendimiento

En esta sección se describen las peculiaridades del problema del fraude a la hora de medir el rendimiento de los modelos así como los ratios y medidas que habitualmente se manejan para dicho problema de clasificación.

La evaluación de los modelos se realiza, como en cualquier problema de clasificación, sobre un conjunto de test independiente de los datos de entrenamiento, pero es necesario tener en cuenta dos aspectos:

- Es necesario que pase un cierto período de tiempo para que una transacción sea considerada no fraude ya que en muchas ocasiones el cliente sobre el que se hace fraude no lo advierte hasta pasado un tiempo. Por tanto, es necesario dejar una ventana de tiempo entre el día que se realiza el test y la última transacción que se considera en él para poder asegurar que todo el fraude ya ha sido marcado. Sin embargo, ha transcurrido tiempo suficiente desde la fecha de operación de los datos utilizados en esta memoria como para no tener en cuenta este aspecto.
- El período de test utilizado es posterior al período usado para entrenar el modelo. Debido a que se trata de un problema que evoluciona en el tiempo y que el modelo es entrenado con el objetivo de la detección de casos de fraude futuros, la forma más realista de medir su rendimiento es tomando un período de tiempo posterior al usado para entrenar y no submuestreando el conjunto total, aproximación habitualmente llevada a cabo en otros problemas de clasificación.

Por otro lado, debido al gran desbalanceo de clases que encontramos en el problema de clasificación de fraude, las medidas de rendimiento utilizadas deben también ajustarse a éste. La figura 3.1 muestra la matriz de confusión correspondiente al problema

de dos clases sobre la que habitualmente son definidas las distintas métricas de rendimiento. En ella, la clase del fraude se asocia con la clase positiva (clase objetivo) y la clase del no fraude con la clase negativa.

A partir de esta tabla (figura 3.1), es habitual medir el rendimiento de un algoritmo de clasificación mediante la precisión (*accuracy*) y el ratio de error, los cuales se definen como

$$acc = (TP + TN)/(P + N) \ ,$$
$$err = (FP + FN)/(P + N) \ .$$

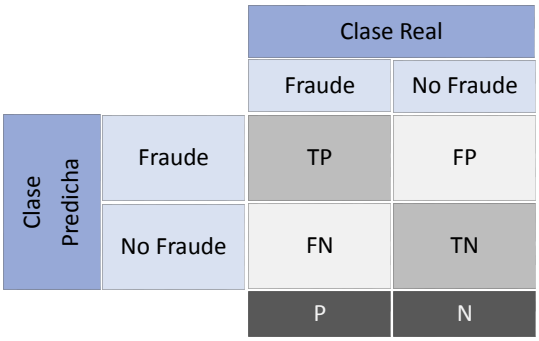

Figura 3.1: Matriz de confusión para el problema de dos clases

En cambio, dichas medidas no son válidas cuando las clases están muy desbalanceadas ya que en este caso los aciertos y/o errores en cada una de las clases no deben considerarse de la misma forma, puesto que el objetivo es la detección de elementos de la clase positiva cometiendo el menor número de errores en la negativa. Un ejemplo que ilustra la poca eficacia de estas medidas es un modelo que clasificase todos los elementos como no fraude; en ese caso obtendríamos una precisión mayor del 99,9 % (y un error menor del 0,1 %) mientras que la utilidad del modelo sería nula. Por tanto, las medidas convencionales de precisión y ratio de error no son adecuadas para este problema.

Dos medidas más apropiadas son el TPR, también conocido como *recall* y el FPR, que se definen como sigue:

- **TPR (True Positive Rate)**. Es la proporción de transacciones de fraude correctamente clasificadas respecto del total de fraude:

$$TPR = TP/P \ .$$

- **FPR (False Positive Rate)**. Es la proporción de transacciones genuinas marcadas como fraude respecto del total de transacciones genuinas:

$$FPR = FP/N \ .$$

Estas dos medidas se combinan en lo que se conoce como curva ROC (*receiver operating characteristic*), la cual representa el TPR en función del FPR, mostrando así las distintas contrapartidas coste (FPR) - beneficio (TPR) que podemos conseguir con el modelo construido. Por tanto, para poder construir dicha curva, es indispensable que la salida del modelo sea dada en forma de un **score de riesgo**, que corresponderá a la probabilidad de pertenencia a la clase del fraude. Así, a cada valor de *score* le corresponderá un cierto FPR y TPR calculados suponiendo que se clasificasen como fraude todas las transacciones con un riesgo mayor o igual que el *score* dado.

A partir de dicha curva, una forma muy extendida de medir el rendimiento global del modelo es calcular el área bajo la curva (AUC), el cual es un valor que oscila entre 0 (todos los ejemplos mal clasificados) y 1 (todos los ejemplos son clasificados correctamente). El área bajo la curva constituye una medida teórica del rendimiento de un modelo pero en la práctica, la elección de un modelo u otro no depende tanto del área cubierta por todos los posibles valores de FPR y TPR sino del nivel de detección conseguida para un cierto nivel de FPR, el cual será definido según las acciones que se vayan a tomar a partir del modelo.

Una limitación añadida en el caso del fraude es que a partir de un cierto número de transacciones alertadas, la entidad financiera no tiene capacidad física de comprobar la autenticidad de todas ellas. Por ello, es un punto clave que el número de transacciones genuinas marcadas como fraude (falsos positivos) no supere un cierto umbral, haciendo irrelevantes todos los ratios conseguidos por encima de éste. En el caso de operaciones con tarjeta, este umbral se encuentra en torno a los 50 puntos básicos (50 transacciones genuinas marcadas como fraude de cada 10 000 operaciones genuinas) mientras que para el caso de transferencias se llega hasta los 200 puntos básicos.

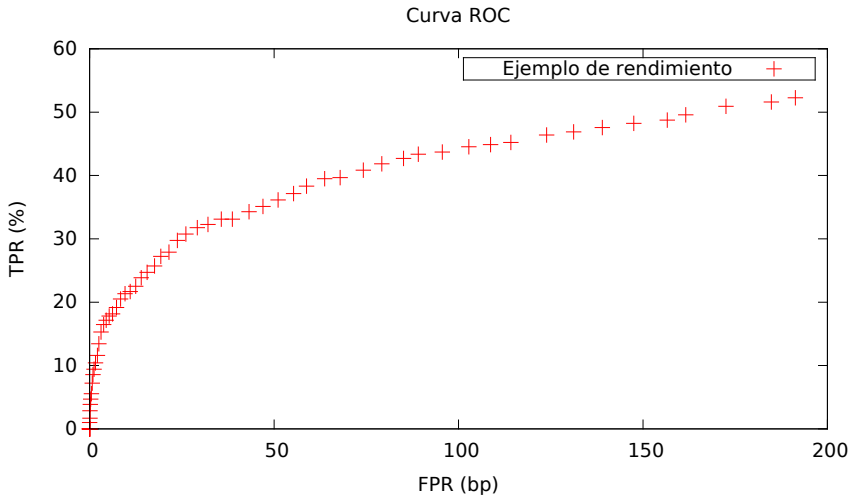

Figura 3.2: Rendimiento TPR en función del FPR (bp). Ejemplo de curva de rendimiento

Debido a las limitaciones dadas por el umbral previamente explicado, las medidas utilizadas para la evaluación de rendimiento son las mismas que las utilizadas comúnmente para definir la curva ROC pero restringiendo el ratio de falsos positivos

hasta el umbral utilizado en la práctica, lo cual hace necesario expresarlo en puntos básicos (tanto por diez mil) para facilitar su representación.

La figura 3.2 muestra un ejemplo del rendimiento obtenido por un modelo de clasificación de fraude representado mediante una curva ROC con las características descritas anteriormente. Los ratios obtenidos en dicha curva corresponden un ejemplo de valores medios esperables para el problema de clasificación de transferencias bancarias.

Como última observación, es interesante remarcar que las medidas realmente utilizadas en la práctica se basan en el importe de las transacciones correctamente clasificadas y no en su número. Esta medida, conocida como VDR (Value Detection Rate), sustituye al TPR utilizado en la curva anterior y corresponde a la proporción de importe de fraude detectado respecto del importe total de las transacciones de fraude. De esta forma, se consigue una curva ROC cuya interpretación es muy intuitiva: dinero de fraude ahorrado (VDR) en función del número de transacciones de cada 10 000 incorrectamente alertadas y que, por consiguiente, implican clientes que han sido molestados al realizar sus compras/transferencias. Al tratarse este estudio de una aproximación teórica al problema de fraude, las medidas comúnmente utilizadas para definir la curva ROC (TPR, FPR) serán las elegidas para evaluar las pruebas del siguiente capítulo (capítulo 4).

Capítulo 4

Experimentos

En este capítulo se explican en detalle los experimentos llevados a cabo para comprobar el rendimiento del algoritmo *random forest* sobre el problema de clasificación de fraude. El algoritmo desarrollado es *Forest-RI* (subsección 2.2.3) ya que, según lo expuesto en los capítulos 2 y 3, es el más adecuado al problema del fraude debido a:

- Sus resultados son muy buenos dentro de los algoritmos basados en conjuntos de clasificadores.
- Es robusto frente al ruido, problema muy presente en la clasificación de fraude.
- La dimensionalidad del problema del fraude es muy alta por lo que no es necesario generar nuevas variables como hace *Forest-RC*.
- Es rápido y fácilmente paralelizable por lo que es idóneo para tratar los grandes volúmenes de datos (tanto en número con en dimensionalidad) que se manejan en el problema de clasificación de fraude.

El capítulo está organizado siguiendo el orden temporal real en el que se fueron desarrollando las pruebas. En primer lugar, se incluyen algunas anotaciones relativas al preprocesado de los datos antes de comenzar con la implementación de los modelos. A continuación y a modo de toma de contacto, se comenzó realizando pruebas a *pequeña* escala utilizando el entorno R. Una vez observados los resultados obtenidos, se implementó el algoritmo en C con el fin de poder procesar cantidades de datos mucho mayores. Por último, se realizan dos experimentos: en primer lugar, se realiza un ajuste de los hiperparámetros del algoritmo y en segundo lugar una comprobación práctica de una medida teórica definida para *random forests*.

4.1. Preprocesado de datos

A continuación se describen las tareas mediante las que, partiendo de las propias transacciones bancarias, se obtienen los dos conjuntos de patrones (entrenamiento y

test) que se utilizarán para construir y validar los modelos. Todas estas tareas son llevadas a cabo tanto para los conjuntos de datos utilizados con la implementación en R (sección 4.2) como para aquellos utilizados con la implementación en C (sección 4.3).

1. **División en conjuntos de entrenamiento y test.** Para el conjunto de entrenamiento se toman al menos seis meses de transacciones (dependiendo del número de datos disponibles), sobre los cuales se realizan los ajustes explicados en el siguiente punto. Un período inmediatamente posterior al considerado en el entrenamiento, debido al carácter temporal del problema del fraude, es utilizado para el conjunto de test.

2. Debido al gran desbalanceo de clases que existe en el problema del fraude, es necesario ajustar los patrones correspondientes a cada una de ellas en el conjunto de entrenamiento. Ninguna modificación se realizará sobre el conjunto de test ya que la evaluación de rendimiento sobre la muestra original constituye la aproximación más ajustada al rendimiento esperado sobre nuevos datos. Con el fin de evitar el desbalanceo de clases en el conjunto de entrenamiento, dos procedimientos son llevados a cabo:

   a) **Submuestreo del no fraude.** En este caso es importante distinguir que no es lo mismo tomar todos los datos de un período de tiempo más corto (por ejemplo partiendo de tres meses de datos para el entrenamiento) que submuestrear los correspondientes a un período de tiempo mayor, aunque se obtenga el mismo número de patrones. Se opta por el submuestreo de un período mayor debido a dos razones: es necesario un período de tiempo más largo para conseguir una muestra representativa de la clase del fraude (ambas clases deben ser tomadas a partir del mismo período) y además, la variedad de patrones de comportamiento será mayor si el período de tiempo considerado es más largo.

   Por ello, se realiza un submuestreo aleatorio simple de un número de patrones adecuado a la características del problema y a la capacidad de procesamiento de la máquina donde se ejecute ya que el submuestreo aleatorio ha demostrado ser una de las técnicas más efectivas de submuestreo sobre la clase mayoritaria cuando la clase minoritaria es la clase objetivo [Japkowicz and Stephen, 2002].

   b) **Sobremuestreo del fraude.** En el caso del sobremuestreo del fraude, se utilizará el algoritmo *SMOTE* [Chawla et al., 2002], el cual genera patrones sintéticos *similares* a los originales, procedimiento que en este caso sí supone una mejora respecto del sobremuestreo aleatorio (mediante replicación de patrones).

   El procedimiento llevado a cabo consiste en el cálculo de los k vecinos próximos de cada patrón. A partir de estos y dependiendo de la proporción que se quiera sobremuestrear, un cierto número de ellos son seleccionados aleatoriamente (si se van a generar un 200 % de ejemplos sintéticos respecto del número de patrones de la clase minoritaria, y k es fijado a 5 vecinos, 2 de

ellos serían elegidos aleatoriamente). Una vez se tienen los vecinos seleccionados, los patrones se generan de la siguiente forma: se calcula el vector de la diferencia entre el ejemplo original y su vecino próximo y se multiplica por un número aleatorio entre 0 y 1; el ejemplo sintético se obtiene sumando el resultado del cálculo anterior al vector del patrón original.

3. **Cálculo de variables.** Una transacción bancaria está formada por distintos campos, los cuales contienen toda la información asociada a la operación (fecha, hora, importe, etc). A partir de los valores de cada operación y de las transacciones previas de esa misma tarjeta/cliente (según se trate de operaciones con tarjeta o de transferencias), se calculan una serie de variables que definen el vector de características correspondiente a cada transacción. Todas las variables calculadas para las pruebas descritas en este capítulo son numéricas con el fin de simplificar la implementación de los árboles (poder restringirlos a este tipo de variables) y también con el de poder usar los mismos conjuntos de datos cuando otros modelos son comparados (por ejemplo, los modelos basados en redes neuronales no pueden manejar variables categóricas).

4. **Tipificación.** Todas las variables son estandarizadas restándoles su media y dividiendo por su desviación típica. Aunque esta transformación no produce ningún efecto cuando el algoritmo utilizado son los árboles de decisión, se consigue así un conjunto de datos que puede ser usado tanto por árboles de decisión como por otros modelos que sí necesitan de la estandarización y con los que se realizarán comparaciones de rendimiento (algoritmos basados en redes neuronales).

4.2. Implementación en R

El problema elegido para esta primera aproximación es la clasificación de operaciones con tarjeta en comercios, ya que como se mencionó en el capítulo anterior, constituye un problema más sencillo. Para su clasificación se ha hecho uso del paquete *randomForest* [Liaw and Wiener, 2002], el cual es una implementación en R del código original Fortran de Leo Breiman y Adam Cutler's.

Dadas las limitaciones del entorno R, los datos que se describen a continuación tienen dimensiones adaptadas a él (tanto en número de datos como en dimensión de las variables) respecto de la dimensionalidad que se maneja en un sistema real de clasificación de fraude.

Los datos seleccionados tienen las siguientes características:

- **Dimensión:** 303 variables. Todas las variables son calculadas en base a la transacción actual y a las transacciones anteriores realizadas por esa misma tarjeta.

- **Conjunto de entrenamiento:** Varios meses entre 2011 y 2012. El submuestreo ha sido ajustado al máximo soportado por R (un conjunto de entrenamiento

mayor de 500 000 provocó errores en ejecución) mientras que con el sobremuestreo se ha multiplicado el tamaño del conjunto de fraude por dos.

  - No fraude: 500 554 casos de no fraude son tomados aleatoriamente de todo el periodo anterior.
  - Fraude: 28 636 casos. Se sobremuestrean los casos de fraude reales generando un caso sintético por cada uno real (el algoritmo se describe en la sección 4.1).

- **Conjunto de test:** Ha sido seleccionado un período de tiempo inmediatamente posterior al utilizado para entrenar. Aunque generalmente todos los casos disponibles son utilizados para realizar el test del modelo, no son procesables por el paquete randomForest todas las operaciones del período elegido, el cual es lo suficientemente grande para que el número de casos de fraude sea significativo. Por tanto, se ha realizado un submuestreo del no fraude ($\sim$ 1 000 000 de datos) mientras que todo el fraude se ha mantenido. El resultado obtenido mediante este procedimiento, si bien no se ajustará exactamente al rendimiento sobre problema real, sí será válido al compararlo con otros modelos bajo las mismas condiciones.

  - No fraude: 999 295 casos de no fraude son tomados aleatoriamente de todo el periodo anterior.
  - Fraude: 1047 casos. Todo el fraude es incluido (sin realizar sobremuestreo) ya que la evaluación de rendimiento debe realizarse sólo sobre casos reales con el fin de ajustarse lo más posible a lo esperado sobre datos nuevos.

Una vez descritos los datos, se describen las dos primeras pruebas realizadas en R, que constituyen una primera toma de contacto con el algoritmo y persiguen dos objetivos:

- Obtener una primera idea del rendimiento del algoritmo *random forest* aplicado al problema del fraude.
- Comprobar que los principales parámetros del algoritmo (el número de árboles a usar y el número de variables a considerar en el corte de cada nodo) se comportan según lo descrito en la subsección 2.2.3: aumentar el número de árboles nunca da lugar a un aumento del error de generalización y aumentar el número de variables consideradas en cada nodo hace aumentar el rendimiento hasta un determinado valor a partir del cual la correlación del conjunto empieza a ser demasiado alta haciendo aumentar el error de generalización.

Con estos dos fines, son construidos 5 *random forests* de 500 árboles cada uno con los datos de entrenamiento anteriormente descritos y en los que cada árbol se construye tomando N elementos con reemplazamiento sobre N, donde N es el número total de ejemplos de entrenamiento (fraude + no fraude, 529 190 en este ejemplo). Lo que distingue a cada uno de estos *random forests* es el número de variables tomadas

aleatoriamente en el corte, parámetro al que le han sido asignados valores en torno a la raíz cuadrada del número de variables (valor recomendado en [Breiman, 2002]). Los valores utilizados han sido los siguientes:

- 2 variables.
- Mitad de la parte entera de la raíz cuadrada de la dimensión. $\left\lfloor \left\lfloor \sqrt{303} \right\rfloor /2 \right\rfloor = 8$
- Parte entera de la raíz cuadrada de la dimensión. $\left\lfloor \sqrt{303} \right\rfloor = 17$
- Doble de la parte entera de la raíz cuadrada de la dimensión. $2 \times \left\lfloor \sqrt{303} \right\rfloor = 34$
- Mitad de la dimensión. $\lfloor 303/2 \rfloor = 152$

La gráfica 4.1 muestra el rendimiento obtenido para cada uno de los cinco modelos sobre el conjunto de test descrito anteriormente (las medidas de rendimiento se describen en detalle en la sección 3.2). La gráfica representa el TPR en tanto por ciento (casos de fraude correctamente clasificados) en función del FPR en puntos básicos (casos genuinos clasificados como fraude), curva que se consigue desplazando el umbral de la proporción de votos a partir del cual un caso es considerado fraude.

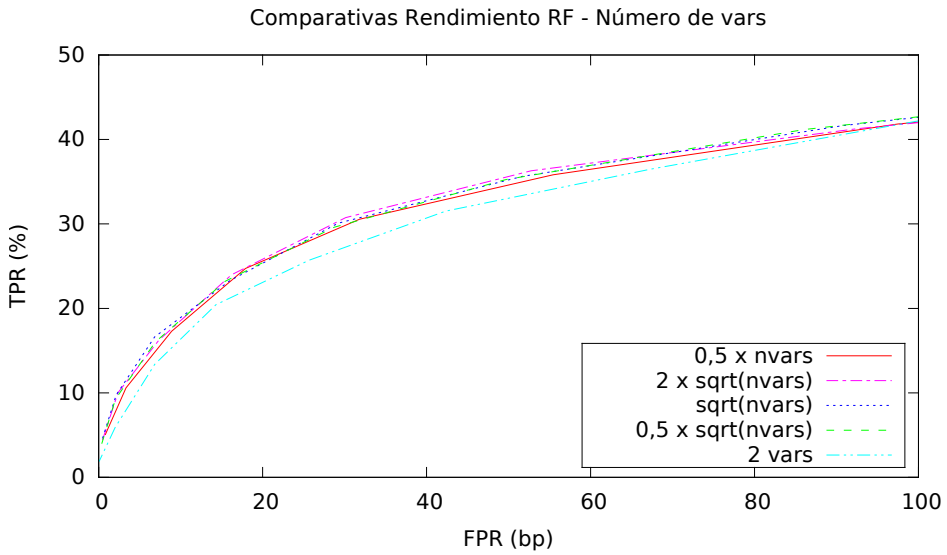

Figura 4.1: Rendimiento TPR en función del FPR (bp). Comparativa valores Num. vars

Por tanto, los puntos más cercanos al origen se conseguirán con un umbral alto: es necesario que un porcentaje muy alto de los árboles considere un caso como fraude para que sea considerado como tal; de esta forma, los errores en la clase del no fraude son bajos pero también lo son los aciertos para la clase del fraude.

De la gráfica anterior (4.1) se extrae la siguiente conclusión:

- El número de variables a considerar en el corte se comporta según lo esperado (subsección 2.2.3) ya que, para valores bajos, aumentar su valor implica un aumento del rendimiento (la fuerza del clasificador aumenta). Sin embargo, a partir de un cierto punto (el doble de la raíz cuadrada en este caso), el rendimiento disminuye ya que la fuerza del clasificador se ha estabilizado mientras que la correlación sigue aumentando, lo cual se traduce en una pérdida de precisión del *random forest*.

La gráfica 4.2 muestra el error OOB (subsección 2.2.1) (en %) para la clase del fraude (clase objetivo) en función del número de árboles. Aunque anteriormente se expuso que el error en la clase del fraude no constituye una buena medida para la evaluación del rendimiento global del problema (sección 3.2), sí constituye una medida válida para observar el punto de convergencia del error en función del número de árboles. Por otro lado, como ya se expuso en la subsección 2.2.1, las estimaciones obtenidas mediante los OOB son tan precisas como las obtenidas en un conjunto de test del mismo tamaño que el conjunto de entrenamiento, por lo que constituyen un buen estimador del error de generalización. Sin embargo, el número de árboles utilizados en el OOB no se corresponde con el real ya que en un *random forest* de T árboles, un ejemplo de entrenamiento sólo es OOB en el 37 % de los árboles. Por ejemplo, si al construir un *random forest* de 100 árboles, el error no estaba aún estabilizado en torno al árbol 37, la estimación OOB no corresponde al error del conjunto total de 100 árboles. Por tanto, los puntos en la gráfica 4.2 donde se alcanza la convergencia usando la estimación OOB no serán tenidos en cuenta de forma estricta. En este caso, se ha representado el error OOB según aumenta el tamaño del *random forest* (y suponiendo el umbral de votación al 50 %).

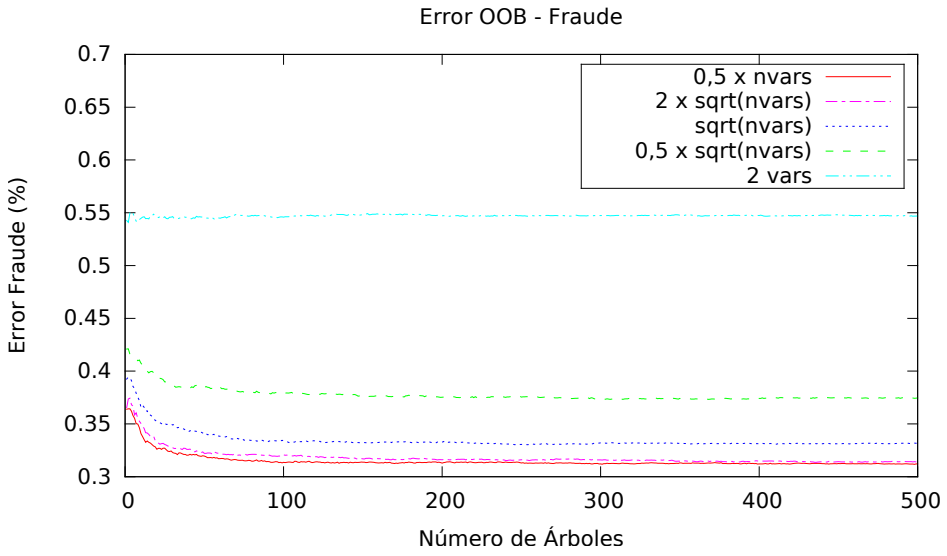

Figura 4.2: Error de clasificación - Clase Fraude. Comparativa valores Num. vars

La conclusión que se extrae de la figura 4.2 es la siguiente:

- A partir de un cierto número de árboles el error se estabiliza. Esto constituye una prueba de que no existe ningún punto a partir del cual aumentar el tamaño del *random forest* hace aumentar el error de generalización (subsección 2.2.3). De esta forma, el modelo nunca se sobreajusta al conjunto de entrenamiento.

Por tanto, se han considerado los siguientes valores para los parámetros del *random forest* para ser comparado con otros algoritmos de clasificación:

- 500 árboles, ya que la estimación OOB del error nos asegura que es un valor en el que el error de generalización está estabilizado.
- El doble de la raíz cuadrada del total de variables, ya que es el valor con rendimiento máximo en la gráfica 4.1.

La gráfica 4.3 evalúa el rendimiento del *random forest* (con 500 árboles y el doble de la raíz cuadrada de la dimensión) comparándolo con otros modelos: LDA (librería *MASS* de R [Venables and Ripley, 2002]), SVM lineal (con algoritmo Pegasos [Shalev-Shwartz et al., 2011]), un solo árbol con y sin poda (librería *rpart* de R [Therneau et al., 2014]) y una red NLDA [Cruz and Dorronsoro, 1998] de una capa y 15 unidades ocultas. Todos los modelos han sido construidos y evaluados con los mismos conjuntos de entrenamiento y test, respectivamente.

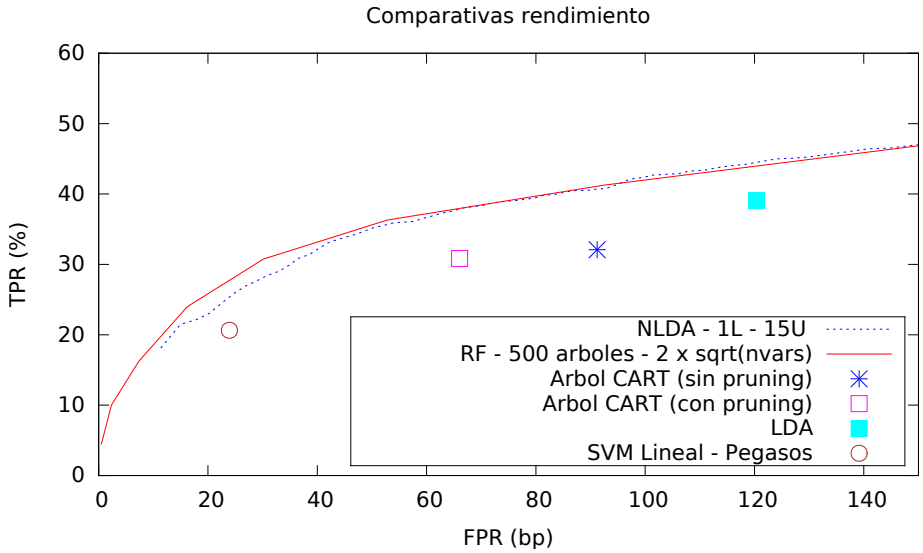

Figura 4.3: Comparativa Rendimientos

Los objetivos y resultados obtenidos con esta comparación son los siquientes:

1. Comprobar que no se trata de un problema lineal. El rendimiento obtenido por los dos modelos lineales (LDA y SVM lineal) es peor que el obtenido por la red NLDA o el *random forest*. Por tanto, como existen modelos no lineales que

mejoran el rendimiento de los algoritmos lineales, podemos afirmar que se trata de un problema no separable linealmente que debe ser atacado con modelos no lineales.

2. Comprobar la mejora de rendimiento que supone el algoritmo *random forest* respecto de un solo árbol de clasificación CART (subsección 2.1.1). El rendimiento obtenido por el *random forest* es significativamente mejor que el obtenido por un solo árbol, tanto cuando se aplica el procedimiento de poda como cuando no. También se observa que, como era de esperar, el efecto de la poda implica una reducción significativa en el número de falsos positivos (de 90 bp a 60 bp) disminuyendo sólo ligeramente el número de ejemplos correctamente clasificados (de 32 % al 30 %).

3. Comparar su rendimiento con el de un algoritmo usado tradicionalmente para la detección del fraude (red NLDA). Se observa que las curvas de rendimiento obtenidas por ambos modelos son muy similares lo cual parece indicar que el algoritmo *random forest* constituye una buena aproximación al problema de clasificación del fraude.

Por tanto, los árboles de clasificación y los modelos lineales quedan descartados como alternativas al *random forest* para el problema de clasificación de fraude mientras que las redes NLDA sí constituyen una buena referencia de rendimiento.

4.3. Implementación en C

Los primeros experimentos utilizando el algoritmo *random forest* sobre una versión reducida (tanto en dimensionalidad como en número de datos) del problema de clasificación del fraude parecen indicar que *random forest* constituye una buena solución al problema planteado. Por ello, el siguiente paso consiste en evaluar su rendimiento sobre un problema a escala real en el que los datos son transferencias bancarias (problema más desbalanceado que el problema de tarjetas expuesto en la sección 4.2).

Para ello, debido a las limitaciones de R para tratar grandes volúmenes de datos, ha sido necesario implementar una **librería en C** del algoritmo *random forest*. Se trata de una librería escalable y preparada para gestionar grandes volúmenes de datos sin limitaciones en cuanto a su número o dimensionalidad, siendo el *hardware* la única limitación existente. Para ello, hace uso de computación paralela (*OpenMP*) y gestión dinámica de memoria. De hecho, ambos elementos constituyen las claves para conseguir una construcción eficiente de los modelos aprovechando al máximo los recursos *hardware* disponibles ya que los experimentos han sido ejecutados en una máquina de 40 *cores* (80 *threads*) y 1TB de memoria RAM.

Una vez implementada la librería, el primer paso es comprobar su funcionamiento. Para ello, han sido utilizados los mismos datos que en la sección anterior (4.2) y tanto su rendimiento como su tiempo de ejecución comparados con el obtenido en R. De esta

forma, se comprueba que el algoritmo está correctamente implementado (comprobando el rendimiento) y se evalúa si la librería en C es más o menos rápida que la de R, la cual realiza llamadas al código Fortran original (a partir de los tiempos de ejecución).

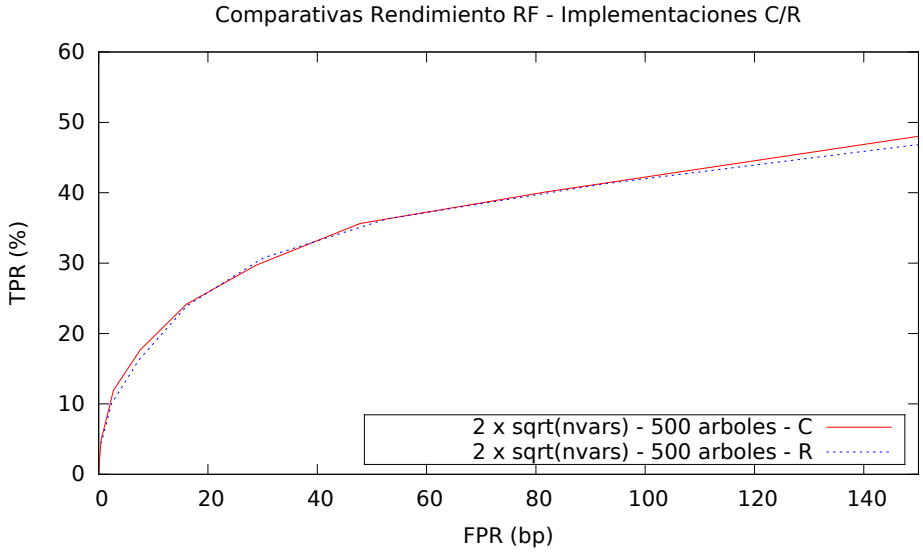

Figura 4.4: Comparativa Rendimientos R/C

La gráfica 4.4 muestra el rendimiento del algoritmo *random forest* para las implementaciones de R y C sobre el problema descrito en la sección 4.2 y los parámetros seleccionados como referencia (500 árboles y el doble de la raíz cuadrada de la dimensión en el corte). Como se puede observar, el resultado es prácticamente el mismo en ambos casos por lo que se puede concluir que el algoritmo en C está correctamente implementado; las pequeñas fluctuaciones pueden deberse a la aleatoriedad propia del algoritmo.

En cuanto a los tiempos de ejecución, han sido medidos utilizando un solo hilo en ambos casos, con el fin de realizar una comparación justa entre ambas implementaciones. La tabla 4.1 muestra los tiempos de construcción y evaluación de cada una de las implementaciones y en ella se ve que la implementación en C es más rápida en ambos casos, lo cual es muy positivo sobre todo en el caso del tiempo de entrenamiento.

	Entrenamiento (por árbol)	Test (RF completo)
R	3 min. 52 s.	17 min. 10 s.
C	1 min. 54 s.	11 min. 7s.

Cuadro 4.1: Comparación tiempos R/C

La conclusión a partir de las comparaciones de ambas implementaciones es que la librería desarrollada es capaz de construir modelos con el mismo rendimiento que la

librería *randomForest* [Liaw and Wiener, 2002] de R en aproximadamente la mitad de tiempo por lo que el entorno R no sólo limitaba el potencial del algoritmo en cuanto al número de datos procesables sino también al tiempo utilizado para construir los modelos. Es importante remarcar que los tiempos son del mismo orden debido a que el paquete en R hace llamadas a código Fortran ya que si se tratase de R puro las diferencias de tiempo entre ambas implementaciones serían mucho mayores. De ahora en adelante, se explotará la implementación multi-hilo ya que el uso de una sola CPU sólo tenía como objetivo la realización de las comparaciones de tiempos de ejecución.

A continuación, en las subsecciones 4.3.1 y 4.3.2, se describen los dos experimentos realizados con la implementación de *random forest* en C. Ambos han sido realizados sobre un conjunto de transferencias bancarias, el cual, como ya se mencionó en la sección 3.1, constituye un problema incluso más desbalanceado que el problema típico de fraude en operaciones con tarjeta. El primer experimento consiste en una búsqueda y selección de los hiperparámetros óptimos del modelo (subsección 4.3.1) y el segundo calcula estimaciones de la fuerza y correlación de distintos *random forests* construidos con distintos parámetros con el fin de conseguir interpretar su comportamiento (subsección 4.3.2).

Los datos utilizados, que en este caso sí se corresponden con los utilizados en un sistema real de detección de fraude, presentan las siguientes características y han sido divididos en los siguientes subconjuntos:

- **Dimensión:** 4355 variables. Todas las variables son calculadas en base a la operación actual y a las transferencias previas realizadas por ese mismo cliente.

  Aunque la dimensionalidad puede parecer muy elevada, la no correlación y la independencia entre los clasificadores son los factores clave a la hora de construir un *random forest*. De esta forma, si se añaden variables relevantes para el problema, la correlación de los árboles disminuirá, ya que hay más variedad a la hora de seleccionar las variables que se evaluarán en cada nodo, sin que los árboles pierdan precisión. En cambio, si se incluyen variables no relevantes para el problema, aunque sean seleccionadas en un determinado nodo, éstas no serán elegidas en el corte, por lo que no deberán afectar ni a la precisión ni a la correlación del conjunto. Este hecho parece indicar que añadir variables no debe empeorar el rendimiento del algoritmo *random forest*.

- **Conjunto de entrenamiento:** Se han utilizado varios meses de datos de entre 2013 y 2014. Como ya se explicó anteriormente (sección 4.1), se ha realizado un submuestreo aleatorio de los patrones de no fraude mientras que el fraude se ha sobremuestrado con el algoritmo *SMOTE* [Chawla et al., 2002]. El mismo conjunto de entrenamiento es utilizado para construir los modelos de los dos experimentos.

  - No fraude: 4 998 311 casos de no fraude son tomados aleatoriamente de todo el periodo de entrenamiento.
  - Fraude: 48 040. Se sobremuestrean los casos de fraude reales generando 5 casos sintéticos por cada uno real.

- **Conjunto de validación:** Corresponde a un período de tiempo inmediatamente posterior al conjunto de entrenamiento. En este caso, todos los patrones (y sólo los reales) son utilizados para realizar la validación. Por tanto, se trata de la proporción real de casos de ambas clases, que corresponde a un caso de fraude por cada aproximadamente 11 500 casos de no fraude. Este conjunto se utilizará en el primer experimento para ajustar los hiperparámetros del modelo y seleccionar los valores óptimos (subsección 4.3.1).
  - No fraude: 8 165 607 casos.
  - Fraude: 701 casos.
- **Conjunto de test:** Se ha seleccionado un período de la misma duración que el conjunto de validación e inmediatamente posterior a él. Al igual que en el conjunto de validación, todos los patrones son considerados en el conjunto de test, el cual se utiliza en los dos experimentos de esta sección. En el primero para comparar el rendimiento del *random forest* (con los parámetros seleccionados a partir del conjunto de validación) con el de una red NLDA y en el segundo para realizar el análisis de fuerza y correlación en los modelos construidos a partir del conjunto de entrenamiento. En este caso, la proporción entre ambas clases es de un caso de fraude por cada aproximadamente 9000 de no fraude.
  - No fraude: 8 067 146.
  - Fraude: 909 casos.

4.3.1. Selección de hiperparámetros

El objetivo de este primer experimento es seleccionar los hiperparámetros del modelo para el problema de fraude en transacciones bancarias. Para ello, se construirán una serie de *random forests* con distintos valores para sus hiperparámetros, cuya combinación óptima se obtendrá a partir de un conjunto de validación. Una vez evaluados los resultados obtenidos, se comprobará su validez sobre un conjunto de test y se compararán con una red NLDA [Cruz and Dorronsoro, 1998].

Se han definido **cinco hiperparámetros** a ajustar en el algoritmo *random forest*. Dos de ellos son los parámetros básicos del modelo (número de árboles y número de variables a seleccionar en cada nodo) mientras que otros tres parámetros han sido añadidos con el fin de aumentar la diversidad entre clasificadores y comprobar si su efecto es el deseado: disminuir la correlación entre árboles manteniendo su fuerza, lo cual se traducirá en una disminución del error de generalización.

Al tratarse de cinco hiperparámetros, realizar una búsqueda en rejilla (prueba de todas las posibles combinaciones) resulta muy costoso computacionalmente, por lo que se ha optado por una optimización de los hiperparámetros mediante búsqueda aleatoria [Bergstra and Bengio, 2012]. De esta forma, una serie de valores son fijados para cada uno de los hiperparámetros y una serie de combinaciones de entre todas las posibles son seleccionadas aleatoriamente. En este caso, se ha optado por la construcción de

cien modelos diferentes ya que se considera una muestra representativa cuyo tiempo de cálculo era viable (la construcción de los cien modelos ha supuesto 27 días de cálculo utilizando 78 hilos por lo que en media, cada uno de los modelos ha tardado aproximadamente 6 horas).

A continuación se describen los hiperparámetros considerados en el ajuste, para los que se han utilizado los valores incluidos en el cuadro 4.2:

- **Hiperparámetros propios del *random forest*.**

  1. **Número de árboles.** El error de generalización de *random forest* se estabiliza a partir de un determinado número de árboles (subsección 2.2.3) por lo que es necesario que el número de árboles utilizado sea mayor que el punto donde el error comienza a estabilizarse. En el otro sentido, tampoco es conveniente utilizar demasiados árboles ya que el tiempo de construcción y evaluación del modelo aumenta al aumentar el tamaño del conjunto. De esta forma, si dos configuraciones obtienen igual rendimiento, se optará por aquellos *random forests* con menos árboles.
  2. **Número de variables seleccionadas en cada nodo.** Aumentar el número de variables seleccionadas hace disminuir el error hasta un determinado punto, a partir del cual la correlación entre los árboles empieza a ser demasiado alta (subsección 2.2.3). Por tanto, es necesario realizar una búsqueda sobre este parámetro para no tomar pocas variables (fuerza baja) o demasiadas (correlación alta). Los valores elegidos están centrados de nuevo en torno a la raíz cuadrada del total de variables.

- **Hiperparámetros enfocados a aumentar la independencia entre clasificadores**. Este conjunto de hiperparámetros constituyen tanto técnicas anteriormente descritas en la parte teórica (sección 2.2) para aumentar la diversidad entre los clasificadores de un conjunto (manipulación de etiquetas de clase) como técnicas que pretenden aumentar la independencia de los clasificadores disminuyendo al mismo tiempo el coste computacional de construcción del algoritmo:

  1. **Número de valores de corte a comprobar en cada variable.** Este parámetro tiene dos finalidades: por un lado, tiene como objetivo la reducción en el tiempo de construcción de los árboles y por otro, busca disminuir su correlación. Indica el número de cortes que se considerarán para cada una de las variables seleccionadas para, en vez de evaluar todos los posibles cortes, solamente comprobar un cierto número de valores n (donde n es el valor del parámetro). Dichos cortes se seleccionan de tal forma que para cualquier intervalo definido por cualquier par de cortes, el número de datos de entrenamiento cuyo valor se encuentra dentro de dicho intervalo, es el mismo. De esta forma, para cada variable, se generan n cortes en función de la distribución de sus valores y así, variando el valor de n, se obtendrán árboles muy poco precisos (si n es pequeña) o muy precisos (si n es grande).

Es una adaptación (mucho menos agresiva en lo que a aleatoriedad se refiere) del método *Extra-Trees* [Geurts et al., 2006], el cual toma aleatoriamente un solo corte para cada una de las variables seleccionadas y escoge el mejor entre todos ellos. Si el valor del parámetro fuese 1, ambos métodos serían equivalentes excepto por el hecho de que mientras el método aquí presentado seleccionaría la mediana de los valores de cada variable como el único candidato de corte, el método *Extra-Trees* lo elegiría aleatoriamente.

2. **Porcentaje de ejemplos de no fraude a utilizar.** Este parámetro tiene también el mismo doble objetivo que el parámetro anterior: reducir la correlación entre clasificadores y reducir el tiempo de entrenamiento. Al disponer de un número tan grande de ejemplos en el caso del no fraude ($\sim 5\,000\,000$), construir árboles a partir de un muestreo *bootstrap* es muy costoso debido a que se seleccionan aproximadamente el 63 % de los datos, por lo que los patrones utilizados en cada árbol estarían en torno a 3 millones. Además, evitar el desbalanceo excesivo dentro de los árboles parece la aproximación correcta (en [Chen et al., 2004] los mejores resultados se obtienen utilizando la misma proporción de ejemplos de fraude que de no fraude).
   Por tanto, si quisiéramos reducir el número de patrones de no fraude utilizados en cada árbol pero mantener el muestreo de N elementos con reemplazamiento (muestreo *bootstrap*), deberíamos reducir los patrones de no fraude y tomar menos de 5 millones. Sin embargo, al eliminar datos de no fraude, estaríamos perdiendo variedad en los datos, la cual hace que los árboles estén menos correlacionados entre sí. Este parámetro produce precisamente este efecto: usar menos elementos de no fraude en cada árbol pero mantener el total de 5 millones de no fraude. Para ello, se calcula el número de patrones de no fraude que corresponden al porcentaje especificado por el parámetro y se realiza un muestreo con reemplazamiento del número obtenido. Por ejemplo, si fijamos el valor del parámetro a un 30 % y partimos de una base de datos de 5 millones, la proporción de datos utilizados (y distintos) sería:

$$P = 1 - \left(1 - \frac{1}{5\,000\,000}\right)^{0{,}3\cdot 5\,000\,000} \simeq 1 - e^{-0{,}3} \simeq 0{,}26\ , \qquad (4.1)$$

   donde el cálculo se ha realizado de la misma forma que el de la muestra *bootstrap* (ec. 2.27). Por tanto, cada árbol utiliza aproximadamente 1 300 000 datos distintos (el 26 % del total de los datos) de no fraude pero el conjunto de datos total utilizado para todo el *random forest* es mucho mayor, y por tanto, más variado, que el que correspondería a un conjunto en el que su muestra *bootstrap* correspondiese a 1 300 000 ejemplos.

3. **Manipulación de las etiquetas de clase.** En este caso, el único objetivo de este parámetro es aumentar la diversidad de los árboles mediante el cambio de las etiquetas de clase. Para ello, se han aplicado las dos técnicas descritas en la sección 2.2: *output flipping* [Breiman, 2000] y *class switching* [Martínez-Muñoz and Suárez, 2005].
   En el caso de dos clases, *output flipping* (OF) intercambia las etiquetas de clase de la mitad de los ejemplos definidos por *flip rate*, es decir, selecciona

aleatoriamente $\frac{flip\ rate}{2} \times N$ patrones de cada clase (donde N es el numero de ejemplos de entrenamiento) y cambia su etiqueta. De esta forma, el valor de *flip rate* siempre debe ser menor que la proporción de elementos que corresponden a la clase minoritaria para intercambiar como mucho la mitad de sus ejemplos y que la clase mayoritaria no *invada* la región del espacio de la clase minoritaria. En el caso del fraude, al estar las clases tan desbalanceadas, el valor de *output flipping* utilizado en estos experimentos corresponde directamente al porcentaje de fraude cuya etiqueta será cambiada, modificando el mismo número de ejemplos en el otro sentido (de no fraude a fraude). Por ejemplo, un valor de OF de 10 en estos experimentos supone intercambiar la etiqueta del 10 % de los casos de fraude y del mismo número de ejemplos de no fraude.

El método *class switching* (CS), por el contrario, asigna a cada patrón una cierta probabilidad p de que su etiqueta sea cambiada por lo que no mantiene la proporción original entre clases tendiendo a conjuntos de entrenamiento más balanceados, siendo, por tanto, el 50 % el valor máximo posible.

Hiperparámetros RF		Hiperparámetros Diversidad		
Árboles	Variables Nodo	Valores Corte	No Fraude (%)	Intercambio Etiquetas (%)
200	32	50	10	0
400	65	100	20	10 OF
600	130	200	30	20 OF
800	260		40	1 CS
999				10 CS
				20 CS

Cuadro 4.2: Valores considerados en el ajuste de los parámetros

La figura 4.5 muestra la curva de rendimiento para las cien combinaciones de hiperparámetros seleccionadas aleatoriamente sobre el conjunto de validación. Al igual que en el experimento realizado con la implementación en R (sección 4.2), las medidas utilizadas son las descritas en la sección 3.2. Las curvas representan el TPR (en tanto por ciento) en función del FPR (en puntos básicos), cuando éste oscila entre 0 y 200, rango utilizado en el problema de clasificación de transferencias bancarias. Como se puede observar, aunque existe un rango de variación entre los rendimientos obtenidos por los distintos modelos (ya que hay algunos de ellos claramente mejores que otros), no existe una sensibilidad excesiva al valor de los parámetros ya que casi todas las curvas se encuentran dentro de un rango aceptable y ninguna combinación

de parámetros (excepto quizás las peores) empeora demasiado el rendimiento obtenido por el algoritmo.

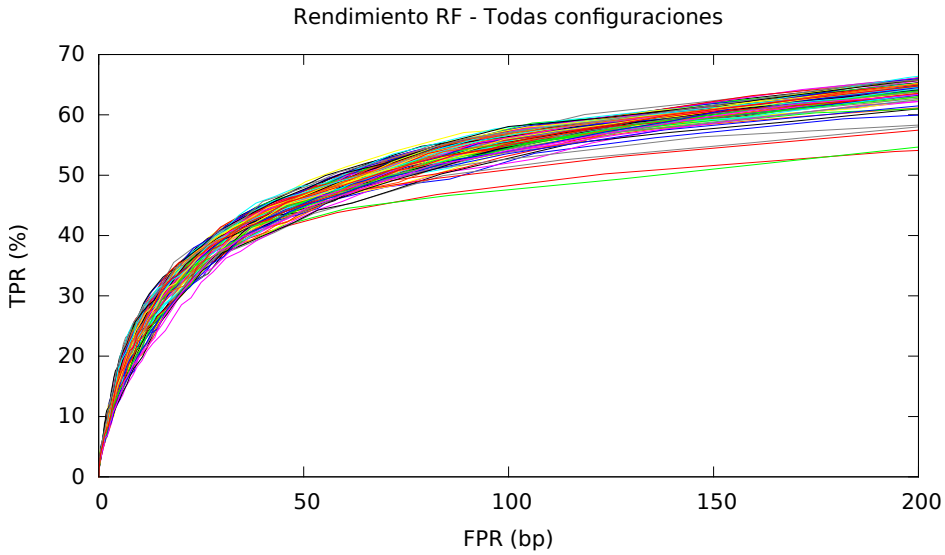

Figura 4.5: Ajuste de parámetros. Todas las configuraciones

Una vez se ha calculado la curva ROC para cada uno de los *random forests* construidos, es interesante identificar qué combinaciones son las que obtienen un mejor rendimiento y cuáles son las que hacen que el algoritmo obtenga peores resultados.

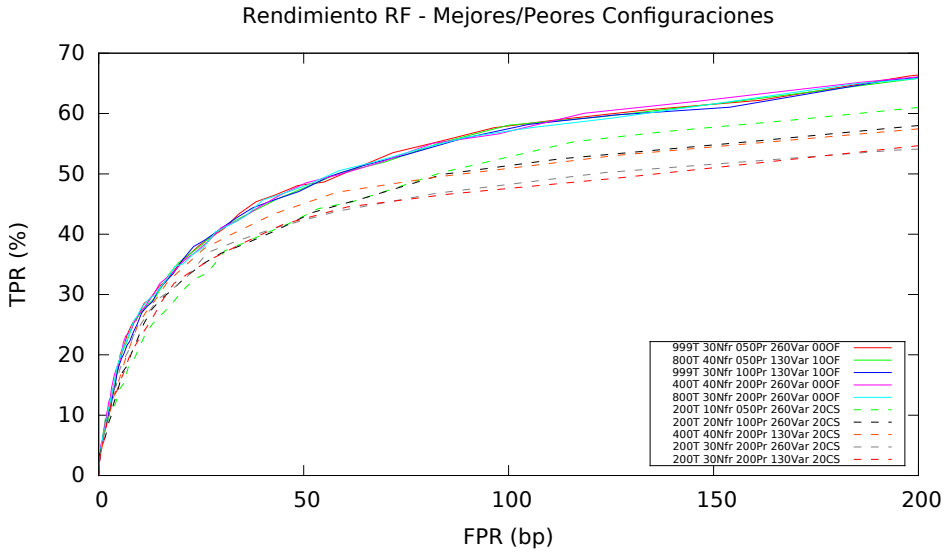

Figura 4.6: Ajuste de parámetros. Mejores/Peores configuraciones. Conjunto de validación.

La figura 4.6 muestra el rendimiento obtenido por los cinco modelos con mayor área bajo la curva y los cinco modelos que cubren las menores áreas sobre el mismo

conjunto de validación. Aunque en la práctica (según se expuso en 3.2) se seleccione un modelo u otro en función del TPR obtenido para un cierto nivel de FPR, en este caso se ha optado por calcular el área bajo la curva para definir el rendimiento de los modelos y poder comparar así las diferentes combinaciones. Sin embargo, sólo ha sido considerado el rango *efectivo* para el problema de clasificación de transferencias (de 0 a 200 puntos básicos) y no todo el área que cubre la curva ROC (de 0 a 10 000 puntos básicos). El cálculo se ha realizado utilizando como método de integración numérica la regla del trapecio (el área bajo cada una de las curvas ha sido aproximada con cinco trapecios). La tabla completa con la ordenación obtenida según este criterio se puede consultar en el apéndice A.

Una vez han sido seleccionados los conjuntos de hiperparámetros con mejores rendimientos en el conjunto de validación, un nuevo conjunto de test ha sido utilizado para confirmar los resultados obtenidos y compararlos con una red NLDA (figura 4.7), cuyos parámetros han sido fijados en base a un estudio previo.

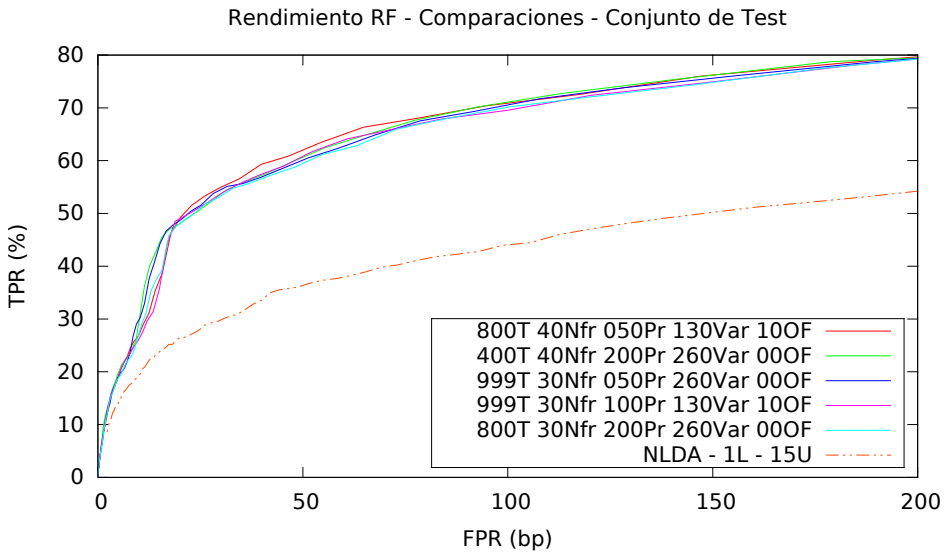

Figura 4.7: Ajuste de parámetros. Mejores configuraciones RF - NLDA. Conjunto de test.

Las conclusiones obtenidas a partir de este experimento son las siguientes:

- Los resultados obtenidos en el conjunto de validación (figura 4.6) se confirman en el conjunto de test (figura 4.7):
  - No existen diferencias entre el rendimiento obtenido por los cinco modelos con mayor área bajo la curva; tanto en el conjunto de validación como en el de test sus cinco curvas de rendimiento son prácticamente iguales. Por tanto, se confirma que *random forest* no es un algoritmo demasiado sensible a los valores de sus parámetros ya que distintas combinaciones de ellos dan lugar al rendimiento máximo y, además, dentro de un rango de valores razonable, no se necesitan ajustes muy finos de los distintos parámetros (no hay una combinación de hiperparámetros mucho mejor que el resto).

  - Aunque los modelos con peores resultados en el conjunto de validación no han sido incluidos en la figura del conjunto de test, su rendimiento sobre éste último se evaluó y la tendencia observada en el conjunto de test fue la misma que la mostrada en el conjunto de validación.
  - Tanto el área bajo la curva como el porcentaje de aciertos para un cierto nivel de FPR es mayor en el conjunto de test que en el de validación. Este comportamiento es intrínseco al problema de fraude, en el que el rendimiento absoluto sobre distintos períodos de tiempo no es comparable ya que la distribución de los datos (así como la proporción de las clases) no se mantiene constante en el tiempo.

- La elección de 100 combinaciones queda justificada, ya no sólo por el tiempo de cálculo, sino también porque el rendimiento obtenido por los mejores modelos es el mismo, lo cual parece marcar una cota del error mínimo posible dentro de los valores evaluados (parece poco probable que una combinación que no ha sido seleccionada aleatoriamente tenga un rendimiento significativamente mejor que el obtenido por las mejores combinaciones).

- Los valores óptimos de los hiperparámetros deben ser considerados de forma conjunta y no cada uno de ellos de forma independiente; sin embargo, se pueden extraer algunas conclusiones sobre ellos:

  - **Número de árboles.** El rango óptimo en cuanto al número de árboles varía de 400 a 1000. Por tanto, existe una cierta combinación de parámetros que hace que 400 árboles sean suficientes para conseguir el error mínimo posible con los hiperparámetros evaluados. Este modelo será preferible sobre los demás debido a que el tiempo necesario para construirlo corresponde a aproximadamente la mitad que el empleado por el resto (las características de los árboles individuales son similares en el resto de modelos y el número de árboles a construir es mucho menor).
  - **Número de variables seleccionadas en cada nodo.** Los resultados parecen indicar que el valor del parámetro que consigue el equilibrio entre fuerza y correlación está en torno a 2-4 veces la raíz cuadrada del total de variables. Un análisis más detallado de este comportamiento se realizará en el siguiente experimento (subsección 4.3.2).
  - **Número de cortes a evaluar para cada variable.** Los tres valores posibles de este parámetro (50, 100, 200) aparecen en las combinaciones óptimas por lo que parece que el rendimiento del algoritmo no se ve muy afectado por él. Sin embargo, todos los valores utilizados hacen reducir de forma muy significativa el tiempo necesario para construir cada uno de los árboles respecto de utilizar todos los posibles valores de corte definidos por los datos.
  - **Porcentaje de ejemplos de no fraude.** Parece que los mayores porcentajes (30/40 %) son los que obtienen mejores rendimientos. Por tanto, las conclusiones de [Chen et al., 2004], en el que se recomienda usar la misma proporción de elementos de ambas clases al construir cada árbol cuando el

problema es desbalanceado, no parecen ser una buena aproximación en el caso del fraude (el desbalanceo en el caso del fraude es mucho mayor que el de los problemas expuestos en [Chen et al., 2004]).

- **Manipulación de las etiquetas de clase.** Ninguna de las dos técnicas (*output flipping* y *class switching*) parecen ser beneficiosas al aplicarse junto a *random forest* en el problema de clasificación de fraude.

- En cuanto a la comparación con la red NLDA, el rendimiento obtenido por el algoritmo *random forest* es significativamente mejor que el conseguido por la red. Aunque no se trata de una comparación del todo justa ya que se podría haber probado *bagging* de redes neuronales, no se ha profundizado en el desarrollo de conjuntos de clasificadores basados en otros algoritmos distintos del árbol de decisión, por quedar fuera del alcance de este trabajo. Esta comparación tiene como único objetivo tener como referencia el rendimiento de otro algoritmo no lineal.

4.3.2. Análisis de fuerza-correlación

Este segundo experimento se centra en uno de los hiperparámetros del modelo: el número de variables seleccionadas aleatoriamente en cada nodo. Mientras que el experimento anterior (subsección 4.3.1) estaba enfocado a nivel práctico (selección de parámetros óptimos de cara a minimizar el error de generalización), en este caso el fin es ilustrar a nivel más teórico cómo distinto número de variables seleccionadas en cada nodo afectan a la fuerza y la correlación del conjunto, las cuales nos permiten caracterizar la independencia y precisión de los árboles individuales.

Además, una vez observado en qué punto se consigue el equilibrio fuerza-correlación, se relaciona con el error de generalización del modelo calculado de la misma forma que en el experimento anterior (a partir del área bajo la curva).

Por tanto, el primer paso es calcular los estimadores de la fuerza y correlación de un *random forest* de T árboles a partir de un conjunto de test, lo cual se realizará de la siguiente forma [Breiman, 2001]:

- **Fuerza.** La fuerza de un conjunto de clasificadores, en el caso de dos clases, se define como (subsección 2.2.3)

$$s = \mathbb{E}_{\mathbf{X},Y}\left[\mathbb{E}_{\Theta}\left[c(\mathbf{X},\Theta) = Y\right] - \mathbb{E}_{\Theta}\left[c(\mathbf{X},\Theta) \neq Y\right]\right] \ . \tag{4.2}$$

  Por tanto, su estimación a partir de un conjunto de test $\mathcal{V}$ es la siguiente:

$$s = \frac{1}{|\mathcal{V}|}\sum_{\mathbf{x}\in\mathcal{V}}\left(\frac{1}{T}\sum_{t=1}^{T} I(c_t(\mathbf{x},\Theta_t) = y) - I(c_t(\mathbf{x},\Theta_t) \neq y)\right) \ . \tag{4.3}$$

  De esta forma, s es la diferencia media, sobre todos los patrones de test, entre la proporción de árboles que clasifican correctamente un patrón y los que lo hacen incorrectamente.

- **Correlación.** La correlación entre los distintos árboles de un *random forest* se puede ver como (subsección 2.2.3):

$$\bar{\rho} = \mathbb{E}_{\Theta,\Theta'}\left[\rho(c(\cdot,\Theta), c(\cdot,\Theta'))\right] \ . \tag{4.4}$$

Sin embargo, a la hora de estimar la correlación a partir del conjunto de test, es conveniente utilizar la siguiente expresión:

$$\bar{\rho} = \frac{\mathrm{var}_{\mathbf{X},Y}(\mathrm{mr}(\mathbf{X},Y))}{(E_\Theta[\mathrm{sd}(\Theta)])^2} \ , \tag{4.5}$$

donde $\mathrm{mr}(\mathbf{X},Y) = \mathbb{E}_\Theta\left[c(\mathbf{X},\Theta) = Y\right] - \mathbb{E}_\Theta\left[c(\mathbf{X},\Theta) \neq Y\right]$. Para poder calcular $\mathrm{var}_{\mathbf{X},Y}(\mathrm{mr})$ y $E_\Theta[\mathrm{sd}(\Theta)]$ se necesita definir el margen de un árbol, que representa la diferencia entre los aciertos y errores de un cierto árbol generado a partir de Θ:

$$\mathrm{rmg}(\Theta,\mathbf{X},Y) = I(c(\mathbf{X},\Theta) = Y) - I(c(\mathbf{X},\Theta) \neq Y) \ . \tag{4.6}$$

De esta forma, $\mathrm{mr}(\mathbf{X},Y)$ es la esperanza sobre $\mathrm{rmg}(\Theta,\mathbf{X},Y)$ con respecto a Θ. Como los vectores Θ utilizados para construir cada árbol son independientes y con la misma distribución, se tiene que:

$$\mathrm{mr}(\mathbf{X},Y)^2 = \mathbb{E}_{\Theta,\Theta'}\left[\mathrm{rmg}(\Theta,\mathbf{X},Y)\cdot\mathrm{rmg}(\Theta',\mathbf{X},Y)\right] \ . \tag{4.7}$$

Y, por tanto, la varianza del margen se puede escribir como:

$$\begin{gathered}\mathrm{var}_{\mathbf{X},Y}(\mathrm{mr}(\mathbf{X},Y)) = \mathbb{E}_{\mathbf{X},Y}\left[\mathrm{mr}(\mathbf{X},Y)^2\right] - \mathbb{E}_{\mathbf{X},Y}\left[\mathrm{mr}(\mathbf{X},Y)\right]^2 = \\ \mathbb{E}_{\Theta,\Theta'}\left[\mathrm{cov}_{\mathbf{X},Y}(\mathrm{rmg}(\Theta,\mathbf{X},Y),\mathrm{rmg}(\Theta',\mathbf{X},Y))\right] = \\ \mathbb{E}_{\Theta,\Theta'}\left[\rho(\Theta,\Theta')\cdot\mathrm{sd}(\Theta)\,\mathrm{sd}(\Theta')\right] \ ,\end{gathered} \tag{4.8}$$

donde $\rho(\Theta,\Theta')$ es la correlación entre $\mathrm{rmg}(\Theta,\mathbf{X},Y)$ y $\mathrm{rmg}(\Theta',\mathbf{X},Y)$, que se interpreta como la correlación entre un cierto par de árboles, y $\mathrm{sd}(\Theta)$ es la desviación típica de $\mathrm{rmg}(\Theta,\mathbf{X},Y)$. La correlación $\bar{\rho}$ se obtiene entonces promediando $\rho(\Theta,\Theta')$ sobre cualquier par de árboles, obteniendo así la expresión inicial (4.5).

Por tanto, para calcular $\bar{\rho}$ a partir de un conjunto de test $\mathcal{V}$ es necesario estimar $\mathrm{var}_{\mathbf{X},Y}(\mathrm{mr}(\mathbf{X},Y))$ y $\mathrm{sd}(\Theta)$, lo cual se hace de la siguiente forma:

$$\mathrm{var}_{\mathbf{X},Y}(\mathrm{mr}(\mathbf{X},Y)) = \frac{1}{|\mathcal{V}|}\sum_{\mathbf{x}\in\mathcal{V}}\left(\frac{1}{T}\sum_{t=1}^{T}\left(I(c_t(\mathbf{x},\Theta_t) = y) - I(c_t(\mathbf{x},\Theta_t)\neq y)\right)\right)^2 - s^2 \ . \tag{4.9}$$

$$E_\Theta[\mathrm{sd}(\Theta)] = \frac{1}{T}\sum_{t=1}^{T}\sqrt{1+(2p_t-1)^2} \ , \tag{4.10}$$

donde

$$p_t = \frac{1}{|\mathcal{V}|}\sum_{\mathbf{x}\in\mathcal{V}} I(c_t(\mathbf{x},\Theta_t) = y) \ . \tag{4.11}$$

Una vez se han obtenido las estimaciones de fuerza y correlación de un cierto *random forest*, se puede definir el **ratio c/s2** [Breiman, 2001], el cual se obtiene dividiendo la correlación por el cuadrado de la fuerza:

$$c/s2 = \frac{\bar{\rho}}{s^2} \ . \tag{4.12}$$

Así, la información de ambos estimadores queda condensada en este cociente, el cual, cuanto más pequeño sea su valor, más preciso será el *random forest* al que corresponda (menor correlación y mayor fuerza).

Por tanto, el siguiente experimento consiste en evaluar el valor del ratio c/s2 para *random forests* en función del número de variables a escoger aleatoriamente en cada nodo. Para ello, se partirá del *random forest* con 400 árboles que tenía rendimiento máximo en el experimento anterior (subsección 4.3.1) y se tomarán los siguientes valores en el número de variables a escoger en cada nodo: 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000 y 3000. Es destacable que el número de nodos de los árboles de cada *random forest* sigue una tendencia decreciente al aumentar el número de variables, desde árboles en torno a 90 000 nodos en el caso de una variable hasta árboles de alrededor de 10 000 al seleccionar 3000 variables. Es el comportamiento esperado ya que, al tener más variables donde elegir, el árbol necesita menos pasos hasta conseguir un nodo puro (nodos hoja).

El cálculo del ratio se ha llevado a cabo tomando aleatoriamente 10 conjuntos de 10 000 datos de no fraude del conjunto de test del experimento anterior junto con todos los datos de fraude para calcular así el ratio medio y su error para cada uno de los *random forests*. A partir de los mismos conjuntos se ha calculado también su área bajo la curva de la misma forma que en el experimento previo.

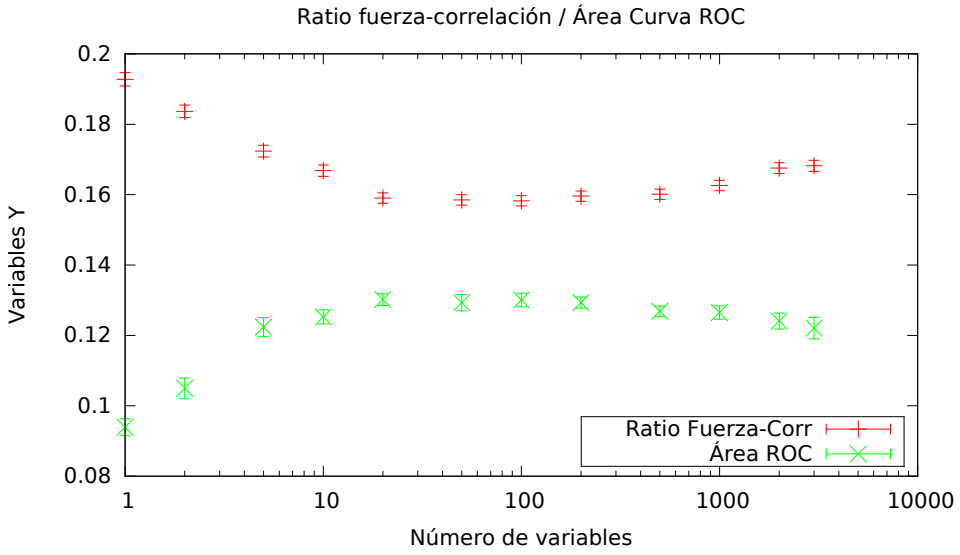

Figura 4.8: Área bajo la curva - Ratio fuerza-correlación según Número de variables

La figura 4.8 muestra los valores obtenidos para cada una de las dos medidas con los

12 *random forests* construidos. Los valores del área han sido reescalados (diviéndolos entre 10 000) para facilitar su representación gráfica.

En la figura 4.8 se observa como los valores óptimos del hiperparámetro analizado se encuentran en el rango de 50 a 200 variables para ambas medidas ya que es en estos *random forests* en los que el ratio se hace mínimo y el área máxima. Como era de esperar, valores muy pequeños dan lugar a *random forests* poco precisos (poco área bajo la curva y valores altos del ratio c/s2), mientras que, por otro lado, valores demasiado altos, hacen también empeorar la precisión del conjunto debido a que la correlación entre árboles aumenta demasiado.

Además, estos resultados coinciden con los valores óptimos obtenidos en el apartado anterior y establecen además una clara relación entre dos medidas obtenidas de forma totalmente distinta: a partir del área sobre la curva ROC y a partir del cálculo del ratio c/s2 definido en [Breiman, 2001]. Por tanto, ambas medidas son equivalentes e igualmente válidas para definir la capacidad de generalización de un *random forest*.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

El principal objetivo de este trabajo ha sido el análisis de la viabilidad de los métodos basados en conjuntos de clasificadores aplicados al problema de clasificación de fraude en medios de pago. En particular, el estudio se ha centrado en el algoritmo *random forest*, el cual presenta resultados muy competitivos dentro de los conseguidos por los conjuntos de clasificadores siendo sencillo, rápido y fácilmente paralelizable.

Con este fin, el trabajo comienza con una revisión detallada de los dos conceptos clave del algoritmo *random forest*: árboles de decisión y conjuntos de clasificadores (capítulo 2). En cuanto a los árboles de decisión, se detallan los procedimientos de construcción, poda y evaluación de árboles centrándose en el algoritmo CART (*Classification and Regression Trees*). En cuanto a los conjuntos de clasificadores, el análisis teórico se ha centrado en entender por qué y bajo qué condiciones este tipo de técnicas consiguen mejoras muy significativas respecto de los clasificadores individuales, obteniendo las siguientes conclusiones:

- El esfuerzo a la hora de construir conjuntos de clasificadores se debe centrar en aumentar la independencia entre los distintos clasificadores que forman el conjunto sin perder demasiado la precisión de cada uno de ellos. Con este fin, existen numerosas técnicas que hacen aumentar la diversidad entre los distintos clasificadores, muchas de ellas basadas en la generación de vectores de números aleatorios.

- La combinación de distintos clasificadores atenúa las restricciones comúnmente encontradas por los clasificadores individuales, como pueden ser datos insuficientes, mínimos locales o espacios de hipótesis equivocados.

- Los conjuntos de clasificadores están orientados a reducir la varianza de los clasificadores individuales por lo que algoritmos con alta varianza son los más adecuados para combinar en un conjunto. Dichos algoritmos son conocidos como

inestables, ya que pequeñas variaciones en el conjunto de entrenamiento pueden provocar grandes cambios en la estructura del modelo.

Además de la visión teórica sobre los conceptos clave del algoritmo *random forest*, se repasan también las peculiaridades del problema a tratar: la clasificación de fraude en transacciones bancarias (capítulo 3). Las principales características de este problema son el gran desbalanceo entre clases, la presencia de ruido en los datos y el gran volumen de estos tanto en número como en dimensión. Dichas características, según lo estudiado en la parte teórica, hacen del algoritmo *random forest* un candidato idóneo con el que realizar los experimentos de la parte práctica del trabajo.

En la última parte del trabajo se realizan una serie de experimentos en los que se aplica el algoritmo *random forest* sobre datos reales de operaciones bancarias (capítulo 4). Los primeros experimentos consisten en una primera aproximación al problema utilizando el entorno R pero, debido al gran volumen y dimensionalidad de los datos, fue necesario el desarrollo de una implementación en C del algoritmo para las siguientes pruebas. Las conclusiones extraídas de esta última parte son las siguientes:

- Se ha comparado el rendimiento obtenido con otros algoritmos de aprendizaje, tanto lineales como no lineales. De las comparaciones se deduce que el problema de clasificación de fraude no es un problema linealmente separable ya que el rendimiento obtenido por los algoritmos no lineales supera al obtenido por los métodos lineales (LDA y SVM lineal). En cuanto a los algoritmos no lineales, se ha observado que *random forest* consigue una mejora de rendimiento significativa respecto de un árbol de decisión así como de una red NLDA.

- El ajuste de sus parámetros no es muy exigente mientras que los valores elegidos se encuentren dentro de un rango razonable: más de 400 árboles, muestreo de un 30-40 % de casos del total de no fraude, como mínimo 50-100 cortes a evaluar en cada nodo, 2-4 veces la raíz cuadrada del total de las variables y sin (o con poco) intercambio de etiquetas de clase. Este amplio margen a la hora de fijar los valores de los parámetros hace del *random forest* un algoritmo muy estable y poco dependiente del valor de sus parámetros.

- Se ha comprobado en los experimentos que las estimaciones de las medidas teóricas de fuerza y correlación de un *random forest* constituyen una referencia del error de generalización equivalente al área bajo la curva ROC, una medida comúnmente usada en la práctica.

En resumen, el algoritmo *random forest* constituye una buena solución al problema de clasificación de fraude en cuanto a rendimiento y a tiempos de construcción y evaluación. Tanto sus niveles de detección, como su robustez frente al ruido en los datos así como su sencilla paralelización lo convierten en una opción muy competitiva para resolver el problema de clasificación de fraude.

5.2. Trabajo Futuro

Este trabajo se ha centrado en obtener una visión en profundidad de los conceptos teóricos sobre los que se apoyan los conjuntos de clasificadores así como en la aplicación práctica del algoritmo *random forest* sobre el problema de clasificación de fraude, ya que, de entre todas las técnicas estudiadas, se ha considerado como la opción más apropiada para abordar el problema del fraude. Sin embargo, hay tres líneas principales sobre las que seguir investigando con el fin de abordar el problema de clasificación de fraude:

1. Preprocesado de datos. La selección de variables y la optimización de los procedimientos de sub/sobremuestreo han quedado fuera del alcance de este estudio por lo que los procedimientos utilizados han sido elegidos en base a la literatura existente. Una revisión de estas técnicas aplicadas sobre el algoritmo *random forest* sería muy adecuada. En concreto, el propio *random forest* ofrece medidas para cuantificar la importancia de las variables (*variable importance*) utilizando los ejemplos *out-of-bag*.

2. Mejoras sobre el algoritmo *random forest*. El desbalanceo de clases puede ser tratado con lo que se conoce como *aprendizaje sensible al coste* (*cost sensitive learning*) [Chen et al., 2004]. Mediante esta técnica, se asigna un determinado peso a cada una de las clases de tal forma que se penaliza más el clasificar mal los elementos de una clase que los de otra. En el caso de los árboles de decisión estos pesos pueden ser utilizados tanto en el cálculo de las impurezas como a la hora de etiquetar un nodo hoja con alguna de las clases.

3. Utilización de otras técnicas basadas en conjuntos de clasificadores. Además del algoritmo *random forest* existen otros métodos basados en conjuntos de clasificadores que sería interesante probar sobre el problema de clasificación del fraude. Por ejemplo, métodos basados en la combinación de clasificadores de distinta naturaleza cuyas predicciones se combinan con un meta-clasificador (*stacking*) o algoritmos basados en *boosting* que presenten mayor estabilidad frente al ruido que *AdaBoost*.

Apéndice A

Selección de hiperparámetros

La siguiente tabla muestra los distintos modelos calculados en el experimento de la sección 4.3.1, ordenados de mayor a menor según el área cubierta bajo la curva en el conjunto de validación.

Árboles	No Fraude (%)	Valores Corte	Variables Nodo	CS/OF (%)	Área
999	30	050	260	0	10445
800	40	050	130	10 OF	10400
999	30	100	130	10 OF	10396
400	40	200	260	0	10377
800	30	200	260	0	10370
200	40	200	260	10 OF	10369
400	40	050	260	20 OF	10361
800	30	050	130	0	10354
800	30	100	130	10 OF	10315
999	40	050	065	10 OF	10296
400	40	050	130	0	10293
600	40	050	032	1 CS	10261
999	30	100	065	20 OF	10258
800	40	200	065	1 CS	10248
800	30	100	065	0	10247
200	30	200	065	1 CS	10244
400	40	200	065	1 CS	10240
800	20	100	260	10 OF	10239
999	30	050	032	20 OF	10231
400	40	100	260	20 OF	10230
400	20	200	130	10 OF	10196

Árboles	No Fraude (%)	Valores Corte	Variables Nodo	CS/OF (%)	Área
999	20	200	260	10 OF	10187
800	40	200	032	10 OF	10184
999	20	050	065	1 CS	10182
400	20	200	260	0	10174
400	40	100	032	20 OF	10170
800	20	050	130	1 CS	10170
600	20	050	130	10 OF	10168
200	20	100	130	0	10167
800	40	050	032	1 CS	10167
999	20	050	032	0	10164
600	20	100	130	20 OF	10154
600	20	050	065	0	10149
999	20	200	065	20 OF	10144
800	20	200	130	1 CS	10135
200	30	050	065	0	10135
400	30	100	065	20 OF	10134
800	30	050	032	10 CS	10128
999	20	100	065	10 OF	10120
400	30	200	065	10 OF	10112
800	20	200	065	0	10106
400	40	100	032	1 CS	10099
999	20	200	260	20 OF	10098
800	20	100	130	1 CS	10083
600	20	200	032	1 CS	10080
200	40	200	130	1 CS	10078
800	30	200	032	10 OF	10067
600	20	100	032	10 OF	10067
200	30	100	065	20 OF	10059
999	20	200	032	1 CS	10056
600	20	050	065	10 OF	10043
200	40	200	032	20 OF	10024
999	30	200	032	10 CS	10024
200	10	100	260	1 CS	10000
200	20	050	065	10 OF	9998
200	40	100	260	1 CS	9991

Árboles	No Fraude (%)	Valores Corte	Variables Nodo	CS/OF (%)	Área
600	10	050	065	1 CS	9976
200	10	050	260	10 OF	9975
400	20	100	130	10 CS	9971
999	10	100	260	20 OF	9964
999	20	200	032	10 CS	9950
600	10	100	260	1 CS	9948
600	10	100	065	10 OF	9922
999	10	100	065	0	9912
600	10	050	130	10 OF	9907
800	10	100	032	20 CS	9900
200	10	050	065	20 OF	9900
800	10	050	065	1 CS	9894
600	30	050	065	20 CS	9891
800	10	050	130	20 OF	9885
600	10	100	130	1 CS	9884
600	30	100	260	20 CS	9874
999	10	050	065	20 OF	9872
800	10	100	260	20 CS	9871
200	30	200	032	1 CS	9870
400	10	200	260	10 CS	9867
200	10	200	130	10 OF	9863
600	10	200	032	10 OF	9856
999	10	200	032	20 OF	9854
999	10	050	032	20 OF	9832
800	10	050	032	10 CS	9829
800	10	050	260	20 OF	9829
800	10	100	130	10 CS	9822
400	30	200	032	10 CS	9819
999	10	100	065	20 CS	9814
999	20	200	130	20 CS	9794
400	10	050	130	10 CS	9788
200	10	050	032	0	9777
999	10	050	032	10 OF	9775
200	20	200	260	10 CS	9723
200	10	100	065	10 CS	9666

Árboles	No Fraude (%)	Valores Corte	Variables Nodo	CS/OF (%)	Área
600	30	100	032	20 CS	9643
200	30	200	065	10 CS	9595
600	40	200	260	20 CS	9565
200	10	200	032	10 CS	9529
200	10	050	260	20 CS	9479
200	20	100	260	20 CS	9239
400	40	200	130	20 CS	9218
200	30	200	260	20 CS	8865
200	30	200	130	20 CS	8807

Cuadro A.1: Rendimientos modelos selección de hiperparámetros

Bibliografía

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305.

Beyer, M. A. and Laney, D. (2012). The importance of 'Big Data': A definition. *Stamford, CT: Gartner*.

Breiman, L. (1996a). Bagging Predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996b). Bias, variance, and arcing classifiers.

Breiman, L. (2000). Randomizing Outputs to Increase Prediction Accuracy. *Machine Learning*, 40(3):229–242.

Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

Breiman, L. (2002). Manual on setting up, using, and understanding random forests v4.0. *Statistics Department University of California Berkeley, CA, USA*.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth International, Canada.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Chen, C., Liaw, A., and Breiman, L. (2004). Using random forest to learn imbalanced data. *University of California, Berkeley*.

Cruz, C. S. and Dorronsoro, J. R. (1998). A nonlinear discriminant algorithm for feature extraction and data classification. *Neural Networks, IEEE Transactions on*, 9(6):1370–1376.

Dietterich, T. G. (2000a). Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer.

Dietterich, T. G. (2000b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification (2nd edition)*. John Wiley.

Freund, Y. and Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer.

Freund, Y. and Shapire, R. (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63(1):3–42.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844.

Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

Martínez-Muñoz, G. (2006). Clasificación mediante conjuntos.

Martínez-Muñoz, G. and Suárez, A. (2005). Switching class labels to generate classification ensembles. *Pattern Recognition*, 38(10):1483–1494.

Mitchell, T. (2006). The discipline of machine learning. Technical Report CMU ML-06 108.

Quinlan, J. R. (1996). Bagging, boosting, and c4. 5. In *AAAI/IAAI, Vol. 1*, pages 725–730.

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30.

Therneau, T., Atkinson, B., and Ripley, B. (2014). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

Whitrow, C., Hand, D. J., Juszczak, P., Weston, D., and Adams, N. M. (2009). Transaction aggregation as a strategy for credit card fraud detection. *Data Mining and Knowledge Discovery*, 18(1):30–55.