Capítulo 31

Breve historia de la familia de clasificadores boosting

Esteban Alfaro Cortes
Matias Gamez Martinez
Noelia Garcia Rubio
Jose Luis Alfaro Navarro
Universidad de Castilla-La Mancha. Albacete
José Mondéjar Jiménez
Universidad de Castilla-La Mancha. Cuenca

Introducción

El aprendizaje automático incluye técnicas de clasificación para aprender a hacer predicciones precisas en base a lo que se ha observado en el pasado. Por ejemplo, para la predicción del fracaso empresarial intentamos construir un clasificador capaz de distinguir las empresas sanas de aquellas que van a fracasar. El planteamiento del aprendizaje automático ante este problema sería el siguiente: Empieza recopilando tantos ejemplos como sea posible de empresas que fracasaron así como de empresas sanas. A continuación con estas observaciones y las etiquetas con sus clases se entrena al sistema de aprendizaje elegido que produce un clasificador o regla de predicción. Posteriormente, ante una nueva empresa de clase desconocida, dicho clasificador intenta predecir si fracasará o no. El objetivo, por supuesto, es generar un clasificador que realice las predicciones más precisas posibles en nuevos ejemplos de test.

Construir un clasificador con una precisión muy alta es en ocasiones una tarea muy difícil. Sin embargo, no es difícil conseguir reglas sencillas con una precisión mucho más moderada. Un ejemplo de este tipo de reglas podría ser algo como lo siguiente: "Si la empresa presenta pérdidas continuadas en los tres últimos años, entonces asignarla al grupo de empresas que va a fracasar". Una regla así ni siquiera cubre todas las empresas, ya que no dice nada de aquellas empresas que no tengan pérdidas continuadas en los tres últimos ejercicios. Sin embargo, las predicciones de esta regla son mejores que las tomadas sólo en base al azar.

Boosting, el método de clasificación que constituye el objeto de este trabajo, se basa en el hecho de que encontrar muchas reglas sencillas puede ser mucho más fácil que encontrar una única regla con una alta precisión para la predicción. Para aplicar el método boosting, empezaremos eligiendo el algoritmo para encontrar las reglas sencillas. El algoritmo boosting entrena a estos clasificadores básicos o débiles repetidas veces, utilizando cada vez distintos conjuntos de entrenamiento, o de forma más precisa, distintas distribuciones o pesos sobre el conjunto de entrenamiento. Cada vez que se le llama, el clasificador básico genera una nueva regla de predicción débil, y después de muchas iteraciones, el algoritmo boosting debe combinar estas reglas débiles en una única regla de predicción que, se espera, sean mucho más precisa que ninguna de las reglas débiles.

Para utilizar este procedimiento hay que tomar dos decisiones fundamentales: primero, cómo debe seleccionarse la distribución en cada iteración, y segundo, cómo deben combinarse las reglas básicas en una sola. Respecto a la elección de la distribución, la técnica utilizada es aumentar el peso de las observaciones que han sido erróneamente clasificadas más a menudo por los clasificadores básicos de las iteraciones anteriores, de esta forma forzamos al clasificador base a centrar su atención en los ejemplos más difíciles. Mientras que para combinar los clasificadores básicos, simplemente tomar el voto mayoritario (ponderado) de sus predicciones es una solución natural y efectiva. Una tercera elección también importante es el método de clasificación que se utiliza para generar los clasificadores sencillos, pero en este trabajo dejaremos abierta esta cuestión ya que se pueden utilizar árboles de clasificación, redes neuronales o cualquier otro, sin que esto sea importante para la descripción general de este método.

Por tanto, Boosting es un método general y probablemente efectivo de producir una regla de predicción precisa combinando reglas sencillas y moderadamente imprecisas de forma similar a la descrita anteriormente. En este trabajo presentamos una revisión histórica de algunos de los trabajos recientes en boosting, centrándonos especialmente en el algoritmo AdaBoost que ha sufrido un intenso estudio teórico y comprobaciones empíricas.

Los orígenes del Boosting

En el aprendizaje supervisado, a partir de un conjunto de entrenamiento dado, el sistema de clasificación generará una regla que asigna a cada observación una de las posibles clases. Esta regla será más o menos precisa dependiendo de la calidad del sistema de clasificación y de la dificultad del problema en cuestión. Intuitivamente, se puede ver que si el clasificador se comporta mejor que la regla por defecto, aunque sólo sea ligeramente, querrá decir que el sistema de clasificación ha encontrado cierta estructura en los datos para lograr su ventaja. Boosting es un método que potencia[1] la precisión del sistema de clasificación aprovechando su ventaja.

Boosting utiliza el sistema de clasificación como una subrutina para generar una regla de predicción, que tiene asegurada una elevada precisión en el conjunto de entrenamiento. Este método aplica el sistema de clasificación en sucesivas ocasiones sobre el conjunto de entrenamiento, pero cada vez centra la atención en ejemplos distintos.

Una vez finalizado el proceso, se combinan los clasificadores básicos en un único clasificador final muy preciso en el conjunto de entrenamiento. Además, este clasificador final, normalmente, es también muy preciso en el conjunto de prueba. Es decir, obtiene

[1] En realidad la traducción literal de Boosting es potenciando.

también notables resultados en la generalización, como han comprobado diversos autores tanto teórica como empíricamente.

Aunque los orígenes pueden remontarse un poco más allá, donde se unen el campo del reconocimiento de patrones y la inteligencia artificial, generalmente se considera que el primer algoritmo de lo que hoy en día ya es una familia de algoritmos Boosting, fue el que propuso Robert E. Schapire en 1990 en su artículo "*The strength of weak learneability*". En ese artículo se propone un algoritmo para potenciar la precisión de lo que se denomina clasificador débil, aquél que es sólo ligeramente superior a la regla del azar o regla por defecto. Este algoritmo es adecuado, en principio, para el caso dicotómico, en el que sólo existen dos clases y se supone que el clasificador C_1 construido sobre el conjunto de entrenamiento T de tamaño n, es sólo ligeramente superior a la regla por defecto, es decir, su error es $\varepsilon = 0{,}5 - \gamma$, donde γ es la ventaja que el clasificador C_1 obtiene sobre la regla por defecto.

Para forzar al sistema de clasificación a aprender más sobre las observaciones difíciles de la distribución, se debe eliminar la ventaja del clasificador C_1 de alguna manera. Schapire propone, a partir del conjunto de entrenamiento T o T_1, construir un nuevo conjunto de entrenamiento T_2, del mismo tamaño n, forzando a que en este nuevo conjunto el clasificador C_1 no sea mejor que la regla por defecto, es decir, que la mitad de los ejemplos en T_2 estén bien clasificados por C_1 pero se equivoque en la otra mitad. Para lograr esto, se lanza una moneda al aire, si sale cara se extraen aleatoriamente ejemplos de T hasta encontrar uno que C_1 sea capaz de clasificar correctamente, es decir, $C_1(x_i) = y_i$. Y si sale cruz, se extraen ejemplos de T hasta que salga uno mal clasificado, o sea, $C_1(x_i) \neq y_i$.

El autor explica que el tiempo de búsqueda es limitado, ya que si la precisión del clasificador C_1 está muy próxima a 0'5, el número de ejemplos extraídos hasta encontrar uno que cumpla con el requisito deseado, estar bien o mal clasificado según salga cara o cruz, sigue una distribución geométrica con probabilidad aproximadamente igual a 0,5. La esperanza de una distribución geométrica de parámetro $p = 0'5$, es $(1-p)/p = 1$. Por tanto, en promedio, se puede esperar que sólo sea necesario extraer dos observaciones, la deseada y otra.

Una vez formado el conjunto de entrenamiento T_2, se entrena de nuevo el sistema de clasificación pero utilizando, en esta ocasión, para ello el conjunto T_2, se obtiene así el clasificador C_2 ,cuyo error debe ser también al menos ligeramente inferior al de la regla por defecto. Por último, se crea un tercer conjunto T_3 , de tamaño n, eliminando de T aquellos ejemplos donde C_1 y C_2 coinciden. Es decir, se extraen observaciones de T hasta encontrar una en la que $C_1(x_i) \neq C_2(x_i)$. Se entrena por tercera vez al sistema de clasificación creando el clasificador C_3.

Para clasificar una nueva observación, x_i, si $C_1(x_i)=C_2(x_i)$ entonces se le asigna la clase acordada por ambos clasificadores, en los demás casos, C_F asigna $C_3(x_i)$. Es decir, en la combinación final C_F se aplica el voto mayoritario de C_1 , C_2 y C_3. Schapire demostró que el error del C_F está acotado por $g(\varepsilon) = 3\varepsilon^2 - 2\varepsilon^3$, que es significativamente menor que el error original ε. Este proceso supone el núcleo fundamental del algoritmo boosting, y se utiliza reiteradamente para mejorar la precisión del clasificador final, llegando a formarse una estructura un tanto complicada que podría reflejarse en la Figura 1.

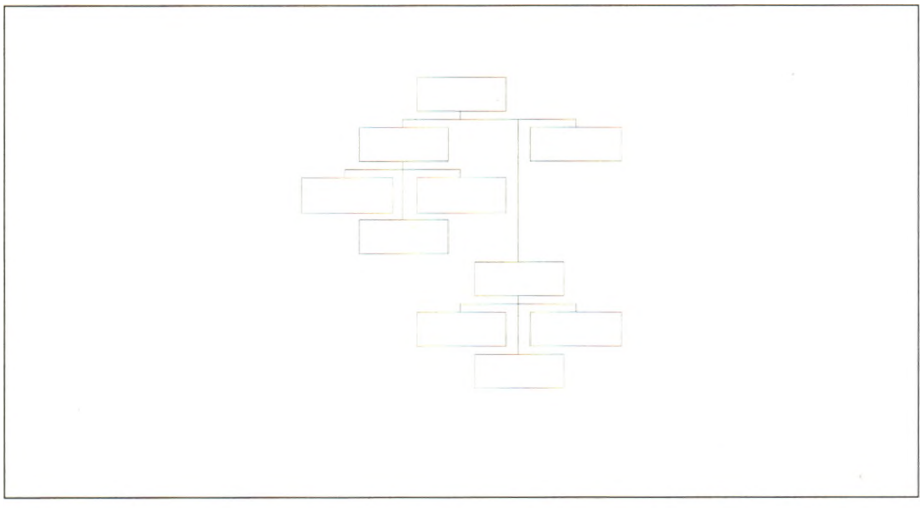

Figura 1: Estructura del algoritmo de Schapire (1990)

A pesar de la importancia que tuvo este primer algoritmo de Schapire su aplicación en la práctica resulta algo complicada. Además, la forma de construir los nuevos conjuntos es poco eficiente, debido a las llamadas recursivas que debe realizar. Poco tiempo después del trabajo de Schapire, Yoav Freund desarrolló un algoritmo más sencillo y eficiente que presentó en su artículo *"Boosting a weak learning algorithm by majority"* Este algoritmo también construye varias distribuciones diferentes sobre el conjunto de observaciones, que se presentan al sistema de clasificación para centrar su atención en las regiones difíciles de la distribución desconocida. El sistema de clasificación genera un clasificador básico para cada distribución que recibe. Por tanto, los clasificadores básicos se comportarán bien en diferentes regiones del espacio de observaciones. El algoritmo boosting combina los clasificadores básicos en un clasificador final mediante el voto mayoritario simple.

En cada iteración, b=1,2, .., B, a los ejemplos del conjunto de entrenamiento se les asigna una distribución de probabilidad en función de la cual se genera el clasificador básico correspondiente a la iteración b-ésima. Aquellos ejemplos clasificados correctamente por ese clasificador reciben una marca. Después de las B iteraciones, los ejemplos que han sido marcados más de $B/2$ veces, son los ejemplos clasificados correctamente por el clasificador final, C_F. Las observaciones restantes, aquellas que han sido marcadas $B/2$ veces o menos[1], serán clasificadas incorrectamente por el clasificador final, el conjunto de estas observaciones se representa por L. Por tanto, el error de C_F en el conjunto de entrenamiento es $|L| / |T|$. El objetivo de este algoritmo boosting es minimizar este error. Para minimizar este error, Freund propone una estrategia de ponderación, que en cada iteración actualiza el peso de la observación x_i en la iteración b-ésima en función de b, B, γ y de cuántas veces x_i ha sido marcada antes de esa iteración. Donde, como ya se ha dicho, γ es la ventaja que los clasificadores básicos deben obtener como mínimo sobre la regla por defecto, siendo su error $\varepsilon_b = 0{,}5\text{-}\gamma$. Para más detalles sobre la estrategia concreta de actualización de los pesos en este algoritmo, véase Freund (1995).

[1] En este caso, se trabaja bajo el supuesto pesimista de que si los empates se rompen de forma aleatoria, la clase elegida es siempre errónea. Por ello, cuando una observación ha sido marcada exactamente $B/2$ veces, se considera que el clasificador final se equivoca.

Lo más importante del algoritmo de Freund es que consigue una estructura más sencilla que la que planteó Schapire originalmente, manteniendo e incluso mejorando la eficiencia, ya que consigue reducir el error de entrenamiento más rápidamente. La Figura 2 ayuda a comprender mejor la diferencia de la estructura de ambos algoritmos.

Figura 2. La parte izquierda del gráfico muestra la estructura compleja del algoritmo de Schapire (1990). A la derecha, la estructura más sencilla de Freund (1995).

Sin embargo, este algoritmo presenta también algunas deficiencias prácticas. En primer lugar, la regla de actualización de los pesos depende de γ, la ventaja que como mínimo deben obtener los clasificadores básicos sobre la regla por defecto. Es decir, esta es la ventaja que obtendrá en el peor de los casos, pero es posible que en realidad se obtenga una ventaja superior en muchos de los casos. En segundo lugar, Freund probó que el algoritmo boosting por mayoría requiere aproximadamente $1/\gamma^2$ iteraciones para reducir el error de entrenamiento hasta cero. Lo que quiere decir que, en el caso extremo de que γ=0,001, se necesitarán un millón de iteraciones. Durante las B iteraciones del proceso, algunos de los clasificadores básicos obtendrán un error menor que 0,5 - γ, pero este algoritmo no es capaz de aprovechar esta ventaja para acelerar el proceso.

Poco tiempo después, Freund y Schapire desarrollaron el algoritmo boosting más utilizado, el AdaBoost (Adaptative boosting), que presentaron en el artículo *"Experiments with a new Boosting Algorithm"* publicado en 1996. En este caso, en la iteración b-ésima se aumenta el peso de los ejemplos mal clasificados por el clasificador básico correspondiente a esa iteración. Además, la regla de actualización de los pesos ya no depende de la constante γ, sino del error obtenido por el clasificador básico en esa iteración.

El comportamiento general de AdaBoost es similar al de los otros algoritmos boosting, puesto que los clasificadores básicos se generan de forma sucesiva y se centra la atención en los ejemplos que resultan más difíciles. La principal diferencia entre AdaBoost y el algoritmo de boosting por mayoría es la regla de actualización de los pesos. AdaBoost utiliza una regla de actualización que depende del error del clasificador básico actual, no de la ventaja que deben conseguir como mínimo, γ. Es decir, en AdaBoost ese factor de la regla de actualización varía en cada iteración, mientras que en el otro caso permanece constante.

Además, otra diferencia es que a cada clasificador básico se le atribuye un peso α_b en función del error que comete. Este peso se utiliza en la combinación final, puesto que AdaBoost utiliza el voto mayoritario ponderado de los clasificadores básicos, en lugar del

voto mayoritario simple. De esta forma, aquellos clasificadores básicos que consiguen una mayor precisión tendrán una mayor importancia en la combinación final.

El algoritmo AdaBoost ha facilitado la aplicación del método boosting y desde su aparición han sido mucho los trabajos realizados al respecto, tanto empíricos comprobando sus buenos resultados, como teóricos, intentando explicar por qué se obtienen tan buenos resultados. Como resultado de esta extensa labor investigadora han surgido diversas modificaciones a este algoritmo, algunas de las cuales se recogen más adelante en el apartado 4 de este trabajo.

AdaBoost

Aunque existen diversas versiones de algoritmos boosting la más extendida es la que proporcionan Freund y Schapire (1996) que se conoce como AdaBoost. Para simplificar se puede suponer que sólo existen dos clases y en el capítulo siguiente se generaliza a más de dos clases. Se parte del conjunto de entrenamiento $T = \{(x_1^b, y_1^b), ..., (x_n^b, y_n^b)\}$ donde Y toma en este caso los valores{-1, 1}. Se asigna a cada observación x_i el peso $\omega_b(i)$, que inicialmente se iguala a 1/n, y posteriormente se irá actualizando en cada iteración. Se construye un clasificador básico a partir del conjunto de entrenamiento ponderado, que se representa por $C_b(x_i)$ y se aplica a cada uno de los ejemplos de entrenamiento. El error de ese clasificador se representa por ε_b y se calcula como

$$\varepsilon_b = \sum_{i=1}^{n} \omega_b(i)\xi_b(i) \qquad \text{donde} \quad \xi_b(i) = \begin{cases} 0 & C_b(x_i) = y_i \\ 1 & C_b(x_i) \neq y_i \end{cases}, \qquad (1)$$

A partir del error del clasificador en la iteración *b*-ésima, se calcula la constante α_b, que se utiliza para la actualización de los pesos. En concreto estos autores hacen $\alpha_b = \ln(1 - \varepsilon_b / \varepsilon_b)$ y el nuevo peso para la iteración *b+1* será

$$\omega_{b+1}(i) = \omega_b(i) \cdot \exp(\alpha_b \xi_b(i)) . \qquad (2)$$

Posteriormente se normalizan los pesos calculados para que la suma de todos ellos sea uno. Según Freund y Schapire, el error debe ser inferior al de la regla por defecto, $\varepsilon_b = 0{,}5 - \gamma_b$, donde γ_b representa la ventaja que obtiene el clasificador básico de la iteración *b*-ésima sobre la regla por defecto, en el peor de los casos en que las dos clases tengan la misma probabilidad a priori, 0'5.

Como se puede ver, aumenta la ponderación de las observaciones mal clasificadas y disminuye la de las clasificadas correctamente, forzando así al clasificador básico construido en la siguiente iteración a centrarse en aquellos casos que han resultado más difíciles. Además, las diferencias en la actualización son mayores cuando el error cometido por el clasificador básico es pequeño, porque si el clasificador consigue una precisión elevada se le da más importancia a los pocos fallos cometidos. Por tanto, la constante alfa puede considerarse como una tasa de aprendizaje calculada en función del error cometido en esa iteración. Además, esta constante también se utiliza en la regla de decisión final, dando más importancia a los clasificadores básicos que cometen un menor error.

Este proceso se repite en todas las iteraciones desde *b* =1, 2, 3, ..,B. Para acabar, se construye el clasificador final como combinación lineal de los clasificadores básicos ponderados por la constante α_b correspondiente. En concreto será:

$$C(x) = sign\left(\sum_{b=1}^{B} \alpha_b C_b(x)\right) \qquad (3)$$

A continuación mostramos El algoritmo AdaBoost

Algoritmo 1. AdaBoost (Freund y Schapire, 1996)

1. Iniciar con $\omega_1(i) = 1/n$, i=1, 2, ..., n.

2. Repetir para b=1, 2, ...,B

   a) Construir el clasificador $C_b(x) \in \{-1,1\}$ utilizando los pesos $\omega_b(i)$ en T^b.

   b) Calcular: $\varepsilon_b = \sum_{i=1}^{n} \omega_b(i)\xi_b(i)$ y $\alpha_b = ln(1 - \varepsilon_b/\varepsilon_b)$

   c) Actualizar los pesos $\omega_{b+1}(i) = \omega_b(i) \cdot \exp(\alpha_b \xi_b(i))$ y normalizarlos.

3. Construir el clasificador final $C(x) = sign\left(\sum_{b=1}^{B} \alpha_b C_b(x)\right)$

AdaBoost puede aplicarse de dos maneras distintas, utilizando remuestreo o utilizando reponderación. En la versión que utiliza remuestreo, se obtiene el conjunto de datos S_b para la iteración b-ésima, mediante una submuestra bootstrap extraída con reemplazamiento, utilizando como probabilidades para la extracción los pesos de las distintas observaciones en esa iteración. En la versión que utiliza la reponderación, el clasificador C_b tiene en cuenta directamente los pesos de los ejemplos. No existe una evidencia fuerte a favor de ninguno de los métodos frente al otro (Breiman,1998), (Freund y Schapire, 1997) y (Freund y Schapire, 1998)

Versiones de AdaBoost

El algoritmo AdaBoost descrito en el apartado 6.4, entrena varias veces al clasificador en el mismo conjunto de entrenamiento, pero centrando su atención en cada paso en los ejemplos que han sido mal clasificados en el paso anterior. AdaBoost se limita al caso dicotómico y utiliza clasificadores básicos cuya salida puede tomar únicamente los valores -1 y 1, por ello se conoce también como AdaBoost discreto, a partir del trabajo de Friedman y otros (2000). Desde su aparición en 1996, han sido varias las modificaciones propuestas para mejorar los algoritmos boosting, en este trabajo se recogen algunas de ellas.

Una generalización del AdaBoost discreto fue propuesta en Freund y Schapire (1996) y explicada en mayor profundidad en Schapire y Singer (1999), esta generalización utiliza predicciones con valores reales, que expresan además el grado de confianza, en lugar de los valores {-1, 1} que utiliza AdaBoost. En este caso, los clasificadores básicos asignan un valor real entre -1 y 1. De tal forma que el signo de $C_b(x)$ informa de la clase predicha y el valor absoluto, $|C_b(x)|$ da una medida de la confianza en la predicción. El valor real de esta

contribución se combina con las anteriores utilizando, como en el algoritmo original, un coeficiente α_b, aunque en esta ocasión se calcula de forma ligeramente distinta.

Friedman(2000) presenta una versión que llama AdaBoost Real, donde los clasificadores básicos estiman la probabilidad de pertenencia a una determinada clase $p_b(x)=P_\omega(Y=1/X=x)\in[0,1]$, donde el subíndice ω indica que esa probabilidad se calcula sobre el conjunto ponderado utilizando los pesos $\omega_b(i)$ correspondientes a cada iteración. La contribución al clasificador final es la mitad de la transformación logit de esta probabilidad estimada.

Algoritmo 2. AdaBoost Real.

1. Iniciar con $\omega_1(i) = 1/n$, $i=1, 2, ..., n$.

2. Repetir para $b=1, 2, ...,B$

   a) Construir un clasificador para obtener una estimación de las probabilidades de las clases $p_m(x) = \hat{P}_\omega(y = 1/x) \in [0,1]$ utilizando los pesos $\omega_b(i)$ en T^b.

   b) Calcular: $f_b(x) = 1/2 \ln(p_m(x)/(1-p_m(x))) \in R$

   c) Actualizar los pesos $\omega_{b+1}(i) = \omega_b(i) \cdot \exp(-y_b f_b(x_i))$ y normalizarlos.

3. Construir el clasificador final $C(x) = sign\left(\sum_{b=1}^{B} f_b(x)\right)$

Como se ha visto en el apartado anterior, el algoritmo AdaBoost minimiza una función de pérdida exponencial. Sin embargo, diversos autores defienden que para clasificación, en lugar de una función de pérdida exponencial, resulta más natural la elección como función de pérdida del logaritmo de la verosimilitud binomial. Basándose en esta función Friedman (2000) propone un nuevo algoritmo que llama Logitboost. Este nuevo algoritmo mantiene la misma filosofía de los anteriores algoritmos boosting, entrenar en sucesivas ocasiones al clasificador forzándolo a centrarse en los ejemplos difíciles, pero la forma de llevarlo a cabo es algo diferente.

En cada iteración se crea una función $f_b(x_i)$ que se ajuste lo mejor posible a z_i^b, que es la respuesta objetivo con la que se trabaja en la iteración b-ésima para la observación i-ésima. Esta respuesta se calcula en función de la verdadera clase de esa observación, que en este caso será $\{0,1\}$, de la probabilidad que se le ha asignado a la clase 1 en la iteración anterior, y del peso en esa iteración.

$$z_i^b = \frac{y_i - p_{b-1}(x_i)}{\omega_i^b} \qquad (4)$$

donde $p_{b-1}(x_i) = P_\omega(Y=1/X=x_i)$ y $\omega_b(i)= p_{b-1}(x_i)\cdot(1-p_{b-1}(x_i))$.

$F_b(x_i)$ es una función de agregación que se va actualizando en cada iteración y se calcula como $F_b(x_i) = F_{b-1}(x_i) + 0{,}5\, f_b(x_i)$ y a partir de ella se calcula la probabilidad de que la observación i-ésima pertenezca a la clase 1 en la iteración b.

$$p_b(x_i) = \frac{\exp(F_b(x_i))}{\exp(F_b(x_i)) + \exp(-F_b(x_i))} = \frac{1}{1+\exp(-2F_b(x_i))} \quad \textbf{(5)}$$

Para comprender mejor este algoritmo cada función de agregación, $F_b(x_i)$, puede considerarse como una estimación de la mitad del logaritmo neperiano del cociente de probabilidades de las clases.

$$F(x) = \frac{1}{2}\ln\left(\frac{p(x)}{1-p(x)}\right) \quad \textbf{(6)}$$

Por tanto, Logitboost ajusta un modelo de regresión logística aditivo mediante optimización secuencial del logaritmo de la verosimilitud binomial, para más detalles véase Friedman (2000).

Una propiedad muy útil de este método es que produce directamente estimaciones de las probabilidades $\hat{P}[Y=1/X=x]$. Esto es muy importante para construir clasificadores cuando los costes de error en la clasificación no son iguales. Además, permite construir clasificadores con la opción de no asignar ninguna clase o asignar la etiqueta "duda" o "no clase" para ciertas observaciones. Una ventaja importante de Logitboost comparado con métodos como las redes neuronales es que funciona bien sin necesidad de un ajuste afinado y sin llevar a cabo una sofisticada optimización no lineal, aunque esta ventaja es general para los algoritmos boosting.

Algoritmo 3. LogitBoost.

1. Iniciar con una función de agregación $F_0(x) \equiv 0$ y probabilidades $p_0(x) \equiv ½$, donde $p(x)$ es la forma abreviada de $\hat{P}[Y=1/X=x]$

2. Repetir para b=1, 2, ...,*B*

a) Calcular la respuesta de trabajo y los pesos para i=1,2, ..,*n*

$$\omega_b(i) = p_{b-1}(x_i)(1-p_{b-1}(x_i)) \; ; \; z_i^b = \frac{y_i - p_{b-1}(x_i)}{\omega_i^b}$$

b) Ajustar una función $f_b(x_i)$ por mínimos cuadrados ponderados

$$f_b(x_i) = \arg\min_f \sum_{i=1}^{n} \omega_b(i)(z_i^b - f_b(x_i))^2$$

c) Actualizar la función de agregación y calcular las nuevas probabilidades.

$$F_b(x_i) = F_{b-1}(x_i) + 0{,}5\, f_b(x_i)$$

$$p_b(x_i) = (1+\exp(-2\, F_b(x_i)))^{-1}$$

3. Construir el clasificador final. $C(x_i) = sign(F_b(x_i))$

Dada la definición de los pesos $\omega(x)$ en aquellos ejemplos donde $p(x)$ esté próximo a 0 ó a 1, $\omega(x)$ llegará a ser muy pequeño. Esto puede causar problemas en el cálculo de $z(x_i)$, por lo que en Friedman(2000) se aconseja tomar las siguientes precauciones:

Si y=1, entonces calcular $z= (y-p)/p(1-p)$ como $1/p$. Como este cociente puede hacerse muy grande si p es pequeño, se debe limitar su valor en una cantidad determinada que se puede representar por *zmax*. El valor concreto elegido para *zmax* no es crucial y Friedman(2000) afirma que valores comprendidos entre 2 y 4 funcionan bien. Por otro lado, si y=0, calcular z como $-1/(1-p)$ con el límite inferior de *-zmax*.

Leo Breiman, autor del método Bagging (Breiman, 1996), mostró inmediatamente un gran interés por el método Boosting que Freund y Schapire propusieron casi simultáneamente a Bagging y que obtenía mejores resultados que éste. El considera que lo fundamental de esta técnica es el uso de remuestreo adaptativo y combinación, y por eso, los llama algoritmos Arcing[1]. La principal diferencia con el método Bagging consiste en entrenar los clasificadores básicos en muestras generadas a partir de distribuciones de probabilidad que van cambiando en función de los errores cometidos, en lugar de permanecer constantes como ocurre en Bagging. Esto implica que los clasificadores básicos utilizados en Bagging son independientes, mientras que en el caso de Boosting cada clasificador depende de los anteriores.

En Breiman (1998) el autor explica que después de probar AdaBoost empezó a sospechar que el éxito de éste no radicaba en su forma concreta sino en el remuestreo adaptativo que realiza, donde se aumentan los pesos de aquellos ejemplos que son clasificados erróneamente con mayor frecuencia. Para comprobarlo, probó tres maneras sencillas de actualizar los pesos o probabilidades. En todos ellos la actualización se realiza en función del valor $1 + m_i^{\ h}$, siendo m_i el número de veces que la observación *i*-ésima ha sido clasificada incorrectamente por los clasificadores básicos construidos hasta esa iteración. Breiman utilizó los valores h=1,2, 4 y este último fue el que mejores resultados le proporcionó. Otra diferencia de arcing respecto a Boosting es que no utiliza ponderaciones en la regla de combinación final, lo que supone también una mayor sencillez del algoritmo al no tener que calcular y guardar esas ponderaciones.

Algoritmo 4. Arcing

1. Iniciar con $\omega_1(i) = 1/n$, i=1, 2, ..., *n*.
2. Repetir para b=1, 2, ...,B
   a) Construir el clasificador $C_b(x)$ utilizando los pesos $\omega_b(i)$ para extraer con reemplazamiento la muestra aleatoria T^b.
   b) Clasificar los ejemplos de T utilizando $C_b(x)$ y calcular $m_b(x_i)$ que es el número de errores cometidos por los *b* primeros clasificadores en la observación i-ésima.
   c) Actualizar los pesos $w_{b+1}(i) = \dfrac{1+m_b(x_i)^4}{\sum_{i=1}^{n} 1+m_b(x_i)^4}$ y normalizarlos.
3. Construir el clasificador final $C(x) = sign\left(\sum_{b=1}^{B} \alpha_b C_b(x)\right)$.

[1] Arcing es el acrónimo del término inglés Adaptive Resampling and Combining. Breiman llama arc-fs al algoritmo AdaBoost, en honor a Freund y Schapire, y a la modificación que él propone arc-x4, pero en este trabajo cuando se hable de arcing se referirá a arc-x4.

Conclusiones

En esta revisión, hemos visto que han surgido muchas interpretaciones o formas de ver de AdaBoost. En primer lugar y más importante, AdaBoost es un algoritmo boosting genuino: teniendo acceso a un verdadero algoritmo de aprendizaje débil que siempre se comporte ligeramente mejor que la regla del azar en todas las distribuciones sobre el conjunto de entrenamiento, podemos probar límites arbitrariamente buenos en el error de entrenamiento y de generalización de AdaBoost.

A partir de la visión original, AdaBoost ha sido interpretado como un procedimiento basado en gradiente descendente funcional o como una aproximación a la regresión logística, entre otras cosas. Todas estas conexiones e interpretaciones han permitido que comprendamos mejor el boosting y han contribuido a su extensión en direcciones incluso más prácticas, tales como regresión logística y otros problemas de minimización de funciones de pérdida, a problemas de más de tres clases, a incorporar regularización y permitir la integración de conocimiento a priori.

Bibliografía

BAUER, E. Y KOHAVI, R. (1999): "An empirical comparison of voting classification algorithm: Bagging, boosting and variants". Machine Learning, vol. 36, pp 105-142.

BREIMAN, L. (1996): "Bagging predictors". Machine Learning, Vol 24, 2, pp.123-140.

BREIMAN, L. (1998). "Arcing classifiers". The Annals of Statistics, Vol 26, 3, pp. 801-849.

DIETTERICH, T.G. (2000): "Ensemble methods in machine learning". En *Multiple Classifier Systems*, Cagliari, Italia.

DRUCKER, H. Y CORTES, C. (1996): "Boosting decision trees". En D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8: Proc. of NIPS'95*, vol. 8, pp. 479-485. The MIT Press.

FREUND, Y. (1995): "Boosting a weak learning algorithm by majority". Information and Computation, 121(2):256-285.

FREUND, Y. Y SCHAPIRE, R.E. (1996): "Experiments with a New Boosting Algorithm". En *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148-156, Morgan Kaufmann.

FREUND, Y. Y SCHAPIRE, R.E. (1997): "A decision-theoretic generalization of on-line learning and an application to boosting". Journal of Computer and System Sciences, 55[1], pp.119-139.

FREUND, Y. Y SCHAPIRE, R.E. (1998): "Discussion of the paper arcing classifiers" de Leo Breiman. The Annals of Statistics, 26[3], pp.824-832.

FREUND, Y. Y SCHAPIRE, R.E., BARTLETT, P. Y LEE, W.S. (1998): "Boosting the margin: A new explanation for the effectiveness of voting methods". The Annals of Statistics, 26(5), pp. 1651-1686.

FRIEDMAN, J.; HASTIE, T. Y TIBSHIRANI, R. (2000): "Additive logistic regression: a statistical view of boosting". The Annals of Statistics, 38(2), pp. 391-293.

KUNCHEVA, L.I. (2004): *Combining pattern classifiers. Methods and algorithms*. Wiley

OPITZ, D. Y MACLIN, R. (1999): "Popular ensemble methods: An empirical study". Journal of Artificial Intelligence Research, 11, pp.169-198.

QUINLAN, J.R. (1996): "Bagging, boosting, and C4.5". En *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pp. 725-730, Menlo Park. AAAI Press / MIT Press.

SCHAPIRE, R.E. (1990): "The strength of weak learnability". Machine Learning, 5(2):197-227.

SCHAPIRE, R.E. Y SINGER, Y. (1999): "Improved boosting algorithms using confidence-rated predictions". Machine Learning, 37(3):297-336.

VALENTINI, G. Y MASULLI, F. (2002): "Ensembles of learning machines". En Marinaro, M. y Tagliaferri, R. (ed) *Neural Nets WIRN Vietri*, Series Lecture Notes in Computer Sciences, Springer-Verlag, Heidelberg, Alemania.