Combinación de clasificadores

Índice

Referencia principal

Cap.7 de "Hands On Machine Learning with Scikit Learn and TensorFlow"
(Todas las imágenes pertenecen a dicha referencia salvo que se indique lo contrario)

Al terminar este tema aprenderemos a construir clasificadores muy potentes a partir de otros que, por si sólos, tienen un rendimiento peor.

1. Intuición

Ya hemos visto una serie de métodos de clasificación, cada uno de ellos con sus características propias. Además desde el principio hemos aprendido que en un proyecto ML se debe evaluar varias veces con conjuntos diferentes antes de implantarlo para que funcione con datos reales. Esta estrategia pretende asegurar que el clasificador sea lo suficientemente complejo para no subajustar los datos con un modelo muy simplista, pero no tan complejo que memorice todos los ejemplos y por tanto no haya aprendido nada. También hemos visto que cada clasificador tiene una expresividad distinta.

Dados varios clasificadores candidatos, sabemos evaluarlos con la curva ROC para determinar cual tiene mejor rendimiento. Pero, a no ser que el área bajo la curva sea igual a 1, no sabemos si aquellos errores cometidos por el mejor son también los que comenten los demás o son distintos. Debido a la expresividad, incluso podría darse el caso de que los errores que cometen diferentes clasificadores sean distintos. Si los pudiéramos combinar, parece lógico pensar que el resultado sería mejor que el de uno sólo.

El término en inglés de para una combinación de métodos es *ensemble*, que añade un significado de cooperación entre los componentes, como un producto ensamblado. En este tema usaremos con frecuencia el término en inglés.

2. Combinación mediante votación

La manera más sencilla e intuitiva de combinar clasificadores es combinar su resultado (evidentemente sobre el mismo problema) mediante una votación. Es decir, la etiqueta asignada a un ejemplo es la etiqueta que más veces ha sido obtenida.

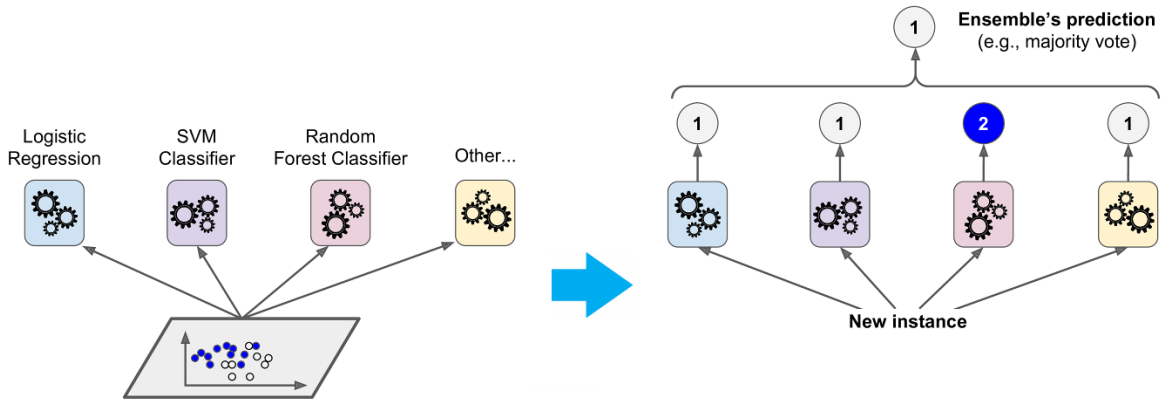

Figura 1: Combinación mediante votación

Estadísticamente, este sistema es capaz de superar el 75 % de precisión, incluso con clasificadores con una área bajo la curva ROC ligeramente superior a 0.5. En la práctica, necesitaríamos muchos clasificadores y de expresividades muy diferentes entre sí, para que esto ocurra.

Si todos los clasificadores involucrados son capaces de devolver, además, la probabilidad de la etiqueta, entonces el ganador de la votación puede mediante una votación ponderada o *soft*. Supongamos que hay tres clasificadores: C_1, C_2 y C_3. Al recibir un nuevo ejemplo cada uno de ellos devuelve una etiqueta $t = \{0, 1\}$ con una cierta probabilidad. Promediamos cada una de las posibles etiquetas y elegimos aquella de mayor probabilidad.

Por ejemplo, en la Tabla 1 se muestran las probabilidades que tres clasificadores asignan a un ejemplo nuevo para dos posibles etiquetas. Con votación *hard* la etiqueta $t = 1$ tiene dos votos y sería ganadora. Sin embargo, con votación *soft* la ganadora (por poco) sería $t = 0$.

Normalmente la votación *soft* se utiliza cuando estamos seguros de que los clasificadores están bien calibrados, es decir tienen un ajuste de sus hiperparámetros correcto.

Tabla 1: Ejemplo de votación *hard* vs. *soft*

	$p(t=0)$	$p(t=1)$
C_1	0.37	0.63
C_2	0.87	0.13
C_3	0.27	0.73
Votos	1	**2**
$\frac{1}{3}\sum_i p_i(t)$	**0.503**	0.496

Finalmente, en la Figura 2 se muestran los resultados obtenidos con 4 clasificadores distintos así como la combinación de estos mediante votación *hard* (o simple) y *soft* (o ponderada).

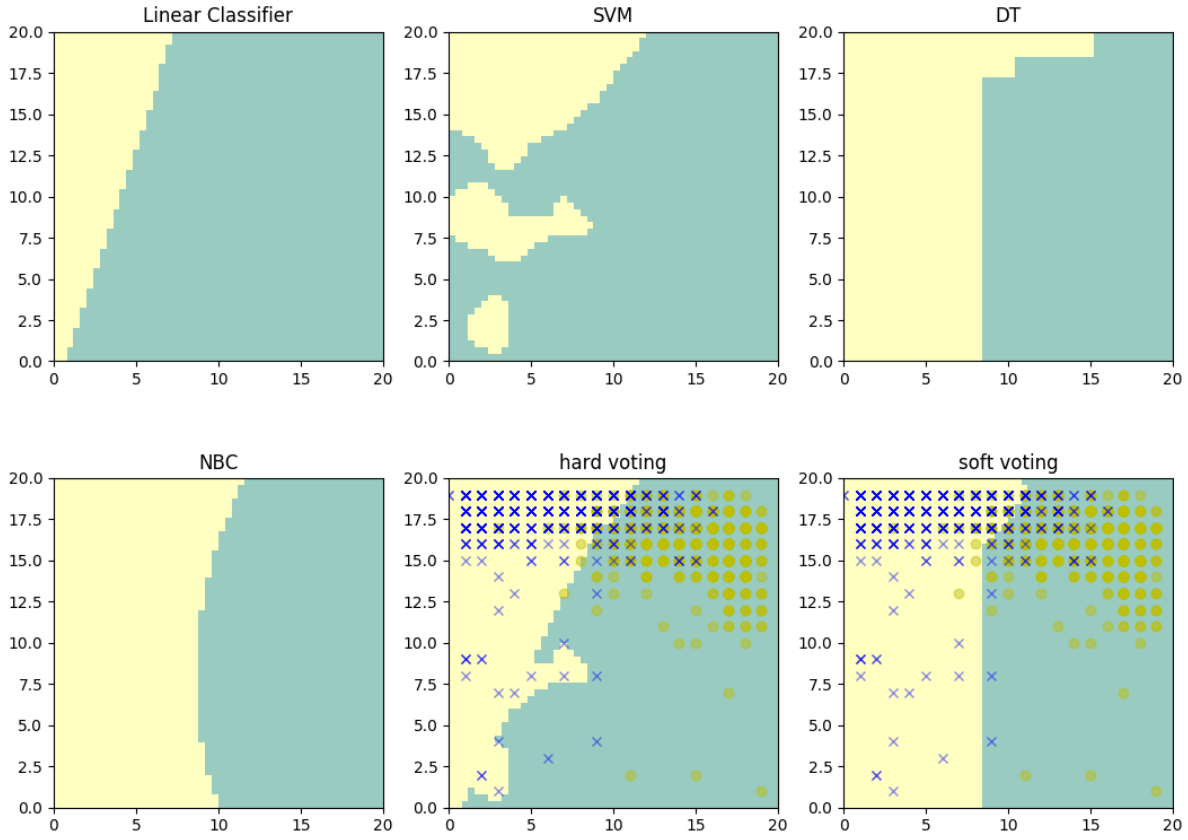

Figura 2: Comparativa de 4 clasificadores individuales (Descenso estocástico Lineal, SVM, DT y NBC) y de su combinación mediante votación *hard* y *soft*. [Fuente: Original de A. Cuesta]

3. Bagging

El término *bagging* es una contracción de *bootstrap* y *aggregation*. *Bootstrap* es el nombre que recibe, en estadística, a remuestrear con remplazo, mientras que *aggregation* es el proceso de combinar datos de diferentes fuentes, tal y como se muestra en la Figura 3

Esta técnica realmente no combina varios clasificadores, sino el resultado de un mismo clasificador sobre diferentes subconjuntos del conjunto de entrenamiento, muestreado con (o sin) remplazo varias veces.

Bagging Con remplazo significa que del conjunto de entrenamiento se crean K subconjuntos formados por N ejemplos aleatorios, aunque cada ejemplo puede ser seleccionado más de una vez durante el proceso de creación. Después se aprende un clasificador con cada uno de ellos.

Pasting Cada ejemplo que pasa a formar parte de un subconjunto no puede volver a ser elegido para ese mismo subconjunto, aunque sí para otro.

Una vez tenemos los K clasificadores entrenados, para estimar la etiqueta de un nuevo ejemplo podemos utilizar alguno de los dos sistemas de votación ya vistos.

Esta técnica es especialmente apropiada para operar sobre grandes conjuntos de datos construyendo subconjuntos más pequeños y trabajando en paralelo. Si el volumen, la velocidad y la variedad de los datos es tal que se puede hablar de "Big Data" con propiedad, entonces se pueden utilizar infraestructuras distribuidas para aprender y estimar basadas en esta técnica. Por ejemplo, recientemente el prof. Jordan y colaboradores de Stanford han presentado *Bag of Little Bootstraps* (BLB) en su artículo "*The Big Data Bootstrap*" del ICML'2012. .

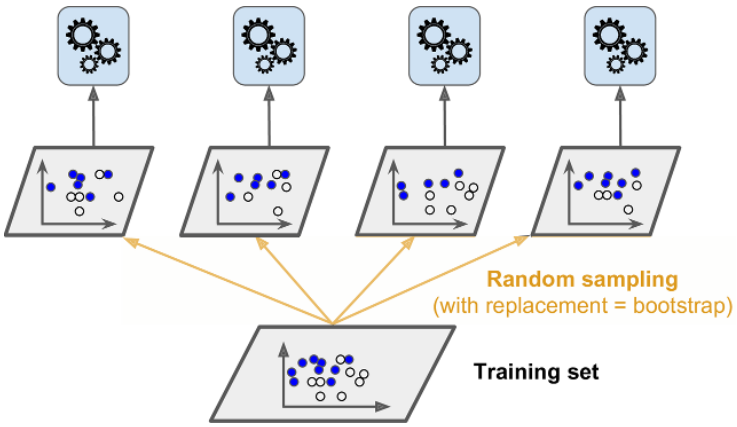

Figura 3: Bagging

Para ilustrar las técnicas con dos ejemplos se incluyen las Figuras 4 y 5. En la Figura 4 se han utilizado 3 métodos diferentes (SVM, DT, NBC) con el mismo número de clasificadores (K) y el mismo número de muestras por clasificadores (N), primero con *bagging* y luego con *pasting*. En la Figura 5 se ha utilizado un solo método (SVM) primero con 10 clasificadores y luego con 100 clasificadores, y el mismo número de muestras en todos los casos. También se ha probado primero con *bagging* y luego con *pasting*.

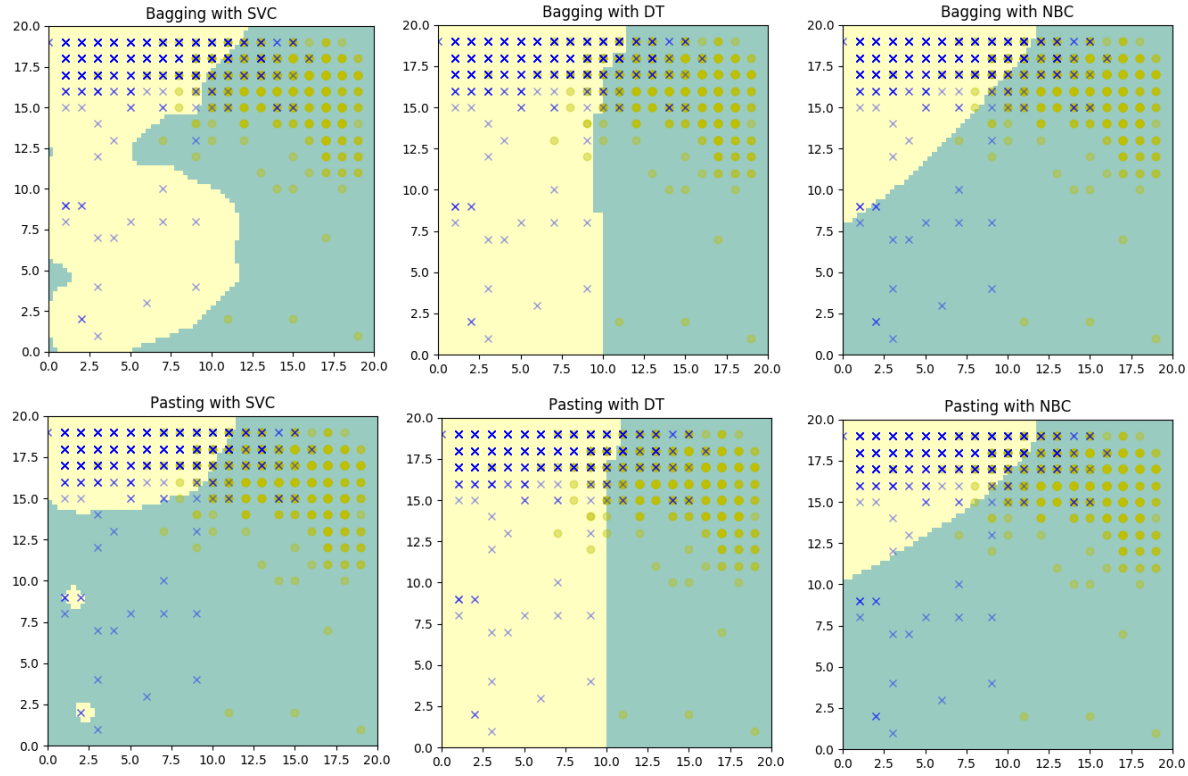

Figura 4: Comparativa de *bagging* vs. *pasting* con 3 clasificadores (SVM, DT y NBC). En todos los casos se utilizaron sólo 20 muestras y 100 clasificadores. [Fuente: Original de A. Cuesta]

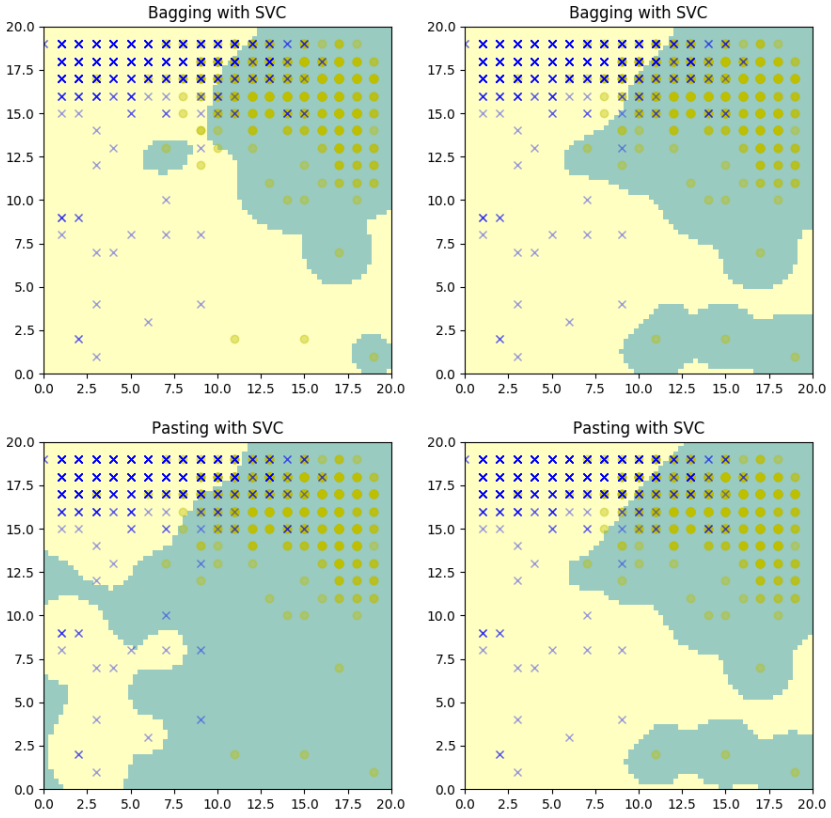

Figura 5: Comparativa de *bagging* (arriba) y *pasting* (abajo) con 10 clasificadores (izq.) y con 100 clasificadores (der.) [Fuente: Original de A. Cuesta]

4. Random Forests

Random forest (RF) esencialmente es un combinación de árboles de decisión (DT), generalmente entrenado con bagging. Esto significa que tiene los hiperparámetros de los DT para controlar el crecimiento del árbol más los hiperparámetros de bagging para controlar la combinación. Pero además, a la hora de crear cada árbol, se seleccionan de manera aleatoria las características que se van a utilizar, con lo que se consigue una gran diversidad de árboles.

Otra característica importante de RF es que proporciona una medida de qué características tienen más "importancia". Generalmente las características con mayor poder discriminador aparecen en nodos próximos a la raíz del árbol, mientras que las que casi no se utilizan para discriminar ejemplos aparecen cerca de las hojas, si es que aparecen.

Por ejemplo, con el conjunto de datos '0' vs. '1', utilizando las 8 características que en su momento construimos a partir de las imágenes, y un RF de 500 árboles, imponiendo una profundidad máxima a estos de 4 niveles, obtenemos el resultado mostrado en la Figura 6. Podemos ver que la característica 0, la 2 y la 3 son las más importantes, según RF.

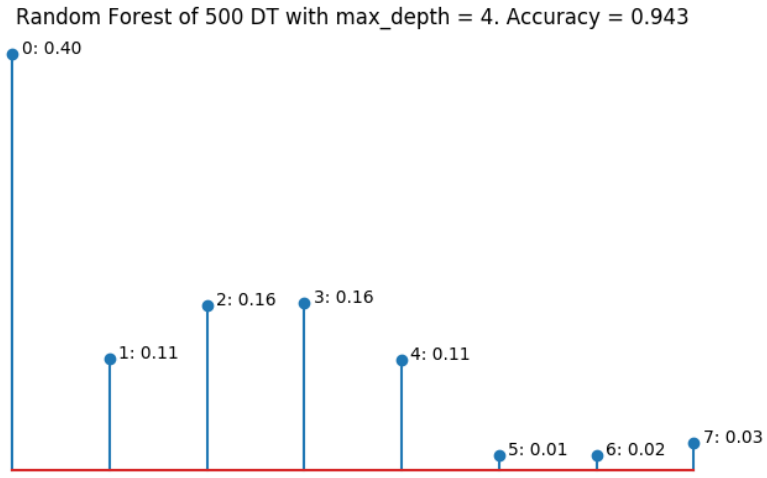

Figura 6: Obtención de la importancia de las características con RF. [Fuente: Original de A. Cuesta]

5. Boosting

El término *boosting* significa "dar un impulso" en el sentido de potenciar algo. La idea de esta técnica es lograr un clasificador "potente" a partir de una secuencia de entrenamientos de uno o varios clasificadores "débiles".

Como el resultado de una iteración es el punto de partida de la siguiente, este algoritmo es muy poco paralelizable. Sin embargo el tiempo de aprender el clasificador débil es muy pequeño, precisamente por ser tan sencillo.

5.1. Adaboost

El término *Adaboost* es una contracción de *Adaptive* y *Boosting*. Se parte de un clasificador muy básico, sencillo y normalmente con muy mal rendimiento. Típicamente es un clasificador que divide en dos el conjunto de entrenamiento atendiendo a unas pocas características, por lo que es muy probable que clasifique mal un número más o menos amplio ejemplos. A continuación se incorpora un nuevo clasificador que "corrige" (mejora, impulsa) al anterior prestando más atención a aquellos ejemplos que fueron mal clasificados; es decir se "adapta".

En la Figura 7 se muestra, cualitativamente, una secuencia de 4 "impulsos". Con el primer clasificador un ejemplo queda mal clasificado (realmente sería mal ajustado). Por tanto el siguiente clasificador "impulsará" los pesos de esos ejemplos mal clasificados para tenerlos más en cuenta y corregir al anterior.

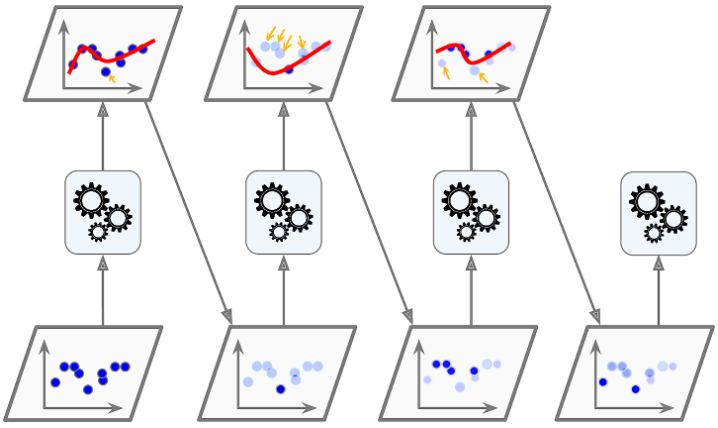

Figura 7: Adaboost

El algoritmo de aprendizaje consta de los siguiente pasos:

1. Comenzamos con el mismo peso para todos los ejemplos del conjunto de datos. Es decir, si tenemos m ejemplos, entonces el i-esimo ejemplo $\mathbf{x}^{(i)}$ tiene peso $w^{(i)} = 1/m$.
2. Realizar sucesivas iteraciones $j = 1, 2, \ldots, K$:
   a) Entrenamos el j-esimo clasificador débil C_j
   b) Calculamos su ratio de error ponderado r_j,

$$r_j = \frac{\sum_{i=1,\ \hat{t}_j^{(i)} \neq t^{(i)}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

   El numerador es la suma de los pesos de todos aquellos ejemplos del conjunto de entrenamiento que el último clasificador entrenado C_j ha clasificado mal.
   El denominador es simplemente la suma de los pesos de todos los ejemplos en esta iteración j.
   c) A continuación se calcula el peso del clasificador C_j como:

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

   De esta manera, si el ratio de error tiende al 100 %, es decir $r = 1$, el peso de dicho clasificador tenderá a $-\infty$, lo cual es lógico ya que parece un clasificador bastante malo. Por otro lado, si el ratio de error tiende al 0 %, o sea $r = 0$, el peso de ese clasificador tan bueno tenderá a $+\infty$. El resultado se puede modular con el hiperparámetro η.
   d) Finalmente actualizamos los pesos de todos los ejemplos mediante la expresión:

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{si } \hat{t}_j^{(i)} = t^{(i)} \\ w^{(i)} \cdot \exp(\alpha_j) & \text{si } \hat{t}_j^{(i)} \neq t^{(i)} \end{cases}$$

   Es decir, los pesos de los ejemplos no clasificados no cambian, y los de aquellos mal clasificados se ven "impulsados" por el valor $\exp(\alpha_j)$.
   e) Es una buena práctica normalizar estos nuevos pesos dividiendo todos ellos por $\sum_{i=1}^{m} w^{(i)}$ en cada iteración para que no crezcan desmesuradamente.

Como resultado de dicho aprendizaje se obtienen K clasificadores, uno por cada iteración. La combinación de todos ellos para estimar la etiqueta de un nuevo ejemplo $\mathbf{z}$ se hace sumando los pesos de todos aquellos clasificadores que otorgan la misma etiqueta para dicho ejemplo. Si por ejemplo hay dos etiquetas (0 y 1), se obtienen dos sumas (la de 0 y la de 1). La etiqueta estimada será la que tenga mayor suma. Formalmente:

$$\hat{t} = \arg\max_{k} \sum_{j=1,\ \hat{t}_j=k}^{K} \alpha_j \quad \text{, para cada posible clase } k.$$

En la Figura 8 se muestra el resultado de Adaboost para árboles de decisión de profundidad 1 y para Naive Bayes. En ambos casos se incrementa el número de clasificadores entrenados desde 9 hasta 90. La Figura 9 muestra el efecto que produce variar el ratio de aprendizaje.

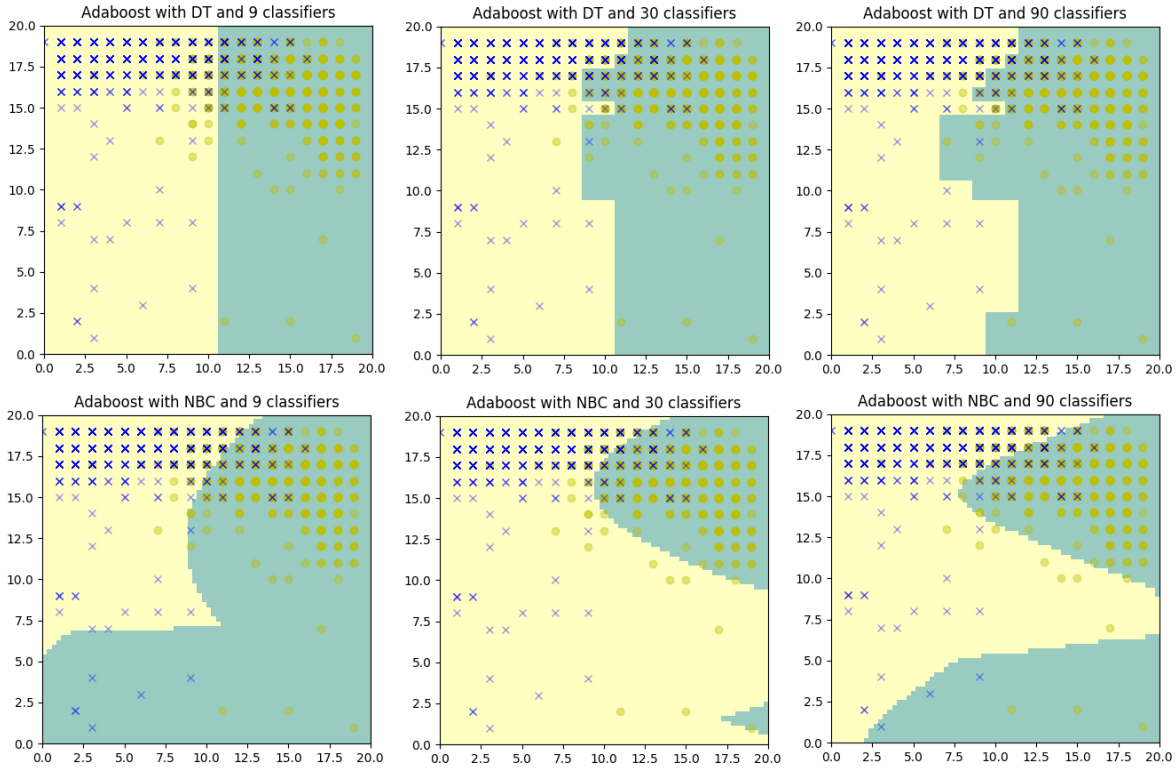

Figura 8: (Arriba) Adaboost con 9, 30 y 90 árboles de decisión de profundidad 1 y (abajo) con 9, 30 y 90 NBCs. [Fuente: Original de A. Cuesta]

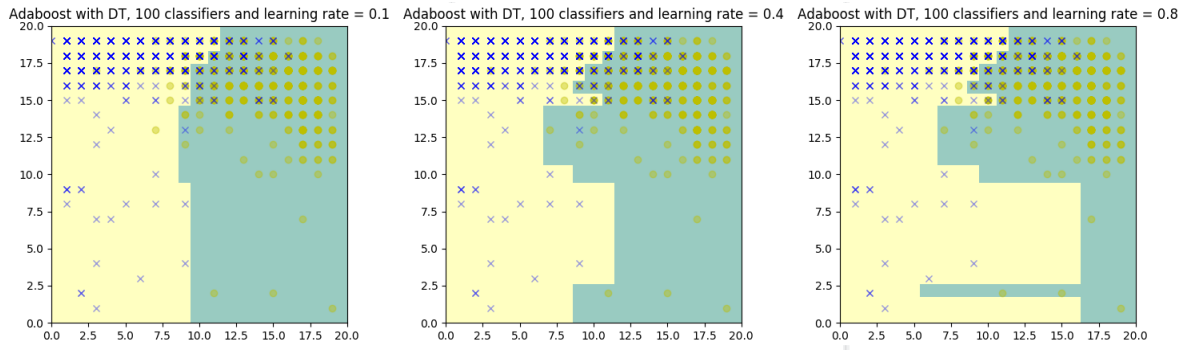

Figura 9: (Arriba) Adaboost con 100 árboles de decisión de profundidad 1 variando el ratio de aprendizaje: (izq.) 0.1, (centro) 0.4, (der.) 0.8. [Fuente: Original de A. Cuesta]

5.2. Gradient Boosting

Este método se diseñó inicialmente para construir árboles de regresión como suma de árboles más "débiles". Es más sencillo de entender de este modo, y ya sabemos desde las primeras semanas que se puede hacer clasificación a partir de modelos de regresión; de modo que lo explicaremos como inicialmente fué concebido.

Un árbol de regresión no es más que un modelo de regresión construido mediante un árbol. Supongamos que existe la función de regresión perfecta (o *golden*) representada por $h^*(\mathbf{X})$, es decir aquella que sería capaz de mapear el conjunto de datos $\mathbf{X}$ al conjunto de valores $\mathbf{y}$ sin cometer ningún error, es decir $h^*(\mathbf{X}) = \mathbf{y}$. El objetivo de *Gradient Boosting* es construirlo, o aproximarlo, del siguiente modo:

$$h^* \simeq h_0 + h_1 + h_2 + h_3 + \cdots + h_K$$

Comenzamos aprendiendo $h_0 = \mathrm{DT}_0$, un árbol de regresión de poca profundidad, sobre el conjunto de entrenamiento $\mathbf{X}$. Es prácticamente imposible que h^* sea precisamente este DT_0, por lo que al ser evaluado sobre el conjunto de datos arrojará errores de estimación. Es decir

$$h_0(\mathbf{X}) = \mathbf{y}_{[0]} \neq \mathbf{y} = h^*(\mathbf{X}) \text{ , donde el subíndice } \cdot_{[0]} \text{ indica que ha sido obtenido con } h_0.$$

Se denominan **residuos** a la diferencia entre el valor verdadero y el valor estimado. En el caso de arriba serían $\mathbf{r}_{[0]} = \mathbf{y} - \mathbf{y}_{[0]}$.

Como nos hemos propuesto reconstruir el árbol de regresión "perfecto" sumando árboles débiles, el siguiente paso sería añadir h_1. El mejor h_1 posible sería aquel con el que lográramos construir $h^* = h_0 + h_1$. En otras palabras, h_1 debería modelar aquello que no logró h_0. Es decir, h_1 mapea el conjunto de datos a los residuos $\mathbf{r}_{[0]}$.

A continuación, igual que antes, es probable que sigamos sin haber logrado reconstruir el árbol "perfecto", y por tanto tendremos de nuevo un residuo, ahora $\mathbf{r}_{[1]}$. Así que añadimos un nuevo árbol débil h_2 que mapea el conjunto de datos a $\mathbf{r}_{[1]}$.

El proceso se resume gráficamente en la Figura 10.

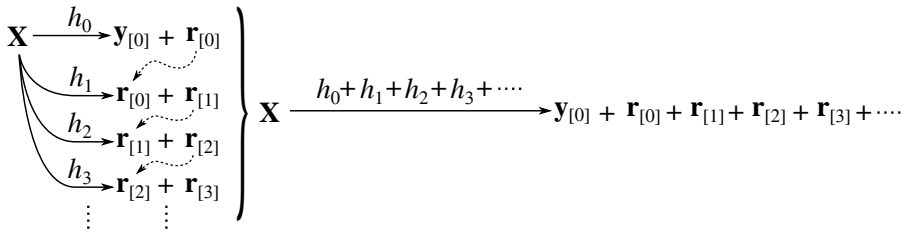

Figura 10: Aprendizaje de árboles de regresión con *gradient boosting*. [Fuente: Original de A. Cuesta]

La regresión es un problema de clasificación con infinitas clases distintas. Por este motivo, para aplicar *gradient boosting* a clasificación tenemos que convertir las clases en un valor continuo. Como ya vimos en su momento, para un problema multi-clase, esto se puede lograr con la función softmax aplicada a la salida para convertir las etiquetas estimadas en probabilidades.

Índice alfabético