Resumen Completo: Boosting y AdaBoost

Elias Sebastian Gill Quintana

1. ¿Qué es Boosting?

Boosting es una familia de algoritmos que convierten **aprendices débiles** (que apenas superan el azar) en **aprendices fuertes** (casi perfectos). La idea surgió de una pregunta teórica: ¿los problemas ”débilmente aprendibles”son iguales a los ”fuertemente aprendibles¿ Schapire [1990] probó que sí, y su demostración construyó el primer algoritmo de boosting.

La idea general es simple: entrenamos varios aprendices secuencialmente, donde cada nuevo aprendiz se enfoca en corregir los errores de los anteriores, y al final combinamos todos para predecir.

2. Procedimiento General de Boosting

```
Input: Distribución de muestra D; Algoritmo base L; Número de rondas T.
Process:
1. D_1 = D.                   # Inicializar distribución
2. for t = 1,...,T:
3.     h_t = L(D_t)           # Entrenar un aprendiz débil
4.     _t = P_{xD_t}(h_t(x)  f(x))  # Evaluar error
5.     D_{t+1} = Adjust_Distribution(D_t, _t)
6. end
Output: H(x) = Combine_Outputs({h_1(x),...,h_t(x)})
```

2.1. Glosario de Símbolos

- D: Distribución de datos original
- L: Algoritmo de aprendizaje base (débil)
- T: Número de rondas de entrenamiento
- h_t: Aprendiz débil entrenado en la ronda t
- ϵ_t: Error de h_t bajo la distribución D_t
- $f(x)$: Función objetivo (etiqueta verdadera)
- $H(x)$: Clasificador final combinado

3. Algoritmo AdaBoost

AdaBoost es la implementación más famosa de boosting. Usa clasificación binaria con clases $\{-1, +1\}$ y minimiza la **pérdida exponencial**:

3.1. Fórmula Fundamental

$$\ell_{\exp}(h \mid D) = \mathbb{E}_{x\sim D}[e^{-f(x)h(x)}]$$

3.2. Glosario

- $\ell_{\exp}$: Función de pérdida exponencial
- $\mathbb{E}$: Valor esperado (promedio)
- e: Número de Euler (aproximadamente 2.718)
- $f(x)$: Etiqueta verdadera (1 o -1)
- $h(x)$: Predicción del clasificador

3.3. Algoritmo Completo AdaBoost

```
Input: Dataset D = {(x_1, y_1), ..., (x_m, y_m)};
       Algoritmo base ; Rondas T.
Process:
1. D_1(x) = 1/m.                    # Inicializar pesos
2. for t = 1,...,T:
3.     h_t = (D, D_t)              # Entrenar clasificador
4.     _t = P_{xD_t}(h_t(x)  f(x)) # Calcular error
5.     if _t > 0.5 then break
6.     _t = (1/2)ln((1-_t)/_t)   # Calcular peso
7.     D_{t+1}(x) = D_t(x)/Z_t × {exp(-_t) si h_t(x)=f(x)
                                    exp(_t)  si h_t(x)f(x)}
8. end
Output: H(x) = sign(_{t=1}^T _t h_t(x))
```

3.4. Explicación Detallada

- **Línea 6**: $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ - Este cálculo le da más peso a los clasificadores que tienen menos error
- **Línea 7**: Actualización de pesos - Aumenta peso de instancias mal clasificadas, disminuye peso de las bien clasificadas
- Z_t: Factor de normalización para que D_{t+1} sea una distribución válida
- sign: Función signo que devuelve +1 o -1

4. Derivación Matemática

4.1. Por qué funciona la pérdida exponencial

Cuando minimizamos la pérdida exponencial, el clasificador resultante alcanza la **tasa de error de Bayes** (el mejor error posible):

$$H(x) = \frac{1}{2}\ln\frac{P(f(x) = 1 \mid x)}{P(f(x) = -1 \mid x)}$$

4.2. Cálculo del peso óptimo α_t

Para encontrar el mejor α_t para cada h_t, resolvemos:

$$\frac{\partial \ell_{\exp}(\alpha_t h_t \mid D_t)}{\partial \alpha_t} = -e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t = 0$$

La solución es exactamente:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

5. Actualización de Distribución

La fórmula de actualización de pesos se deriva como:

$$D_{t+1}(x) = D_t(x) \cdot e^{-f(x)\alpha_t h_t(x)} \frac{\mathbb{E}_{x\sim D}[e^{-f(x)H_{t-1}(x)}]}{\mathbb{E}_{x\sim D}[e^{-f(x)H_t(x)}]}$$

6. Implementación Práctica

- **Re-weighting**: Modificar los pesos de las instancias
- **Re-sampling**: Muestrear según la distribución deseada
- Ambas approaches funcionan similarmente, pero re-sampling permite reiniciar" si un aprendiz es muy malo

7. Análisis Teórico

7.1. Cota de Error

El error del clasificador final está acotado por:

$$\epsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)} \leq e^{-2\sum_{t=1}^{T}\gamma_t^2}$$

Donde $\gamma_t = 0{,}5 - \epsilon_t$ se llama el **edge** o ventaja del aprendiz.

7.2. Cota de Generalización

El error de generalización (en datos nuevos) está acotado por:

$$\epsilon_D \leq \epsilon_D + \tilde{O}\left(\sqrt{\frac{dT}{m}}\right)$$

Donde:

- d: Dimensión VC de los aprendices base (medida de complejidad)
- m: Número de instancias de entrenamiento
- T: Número de rondas
- $\tilde{O}$: Notación O que esconde términos logarítmicos

8. Explicación por Márgenes

8.1. Definición de Margen

El margen de clasificación mide "qué tan segura.es una predicción:

$$f(x)H(x) = \frac{\sum_{t=1}^{T} \alpha_t f(x) h_t(x)}{\sum_{t=1}^{T} \alpha_t}$$

8.2. Cota basada en Márgenes

$$\epsilon_D \leq P_{x\sim D}(f(x)H(x) \leq \theta) + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2} + \ln\frac{1}{\delta}}\right)$$

Esta cota muestra que mientras más grandes sean los márgenes, menor será el error de generalización.

9. Visión Estadística

9.1. LogitBoost

Alternativa que usa pérdida logarítmica en lugar de exponencial:

$$\ell_{\log}(h \mid D) = \mathbb{E}_{x\sim D}\left[\ln\left(1 + e^{-2f(x)h(x)}\right)\right]$$

9.2. Algoritmo LogitBoost

```
Input: Dataset D; Algoritmo de mínimos cuadrados L; Rondas T.
Process:
1. y_0(x) = f(x)
2. H_0(x) = 0
3. for t = 1,...,T:
4.     p_t(x) = 1/(1+e^{-2H_{t-1}(x)})
5.     y_t(x) = (y_t(x)-p_t(x))/(p_t(x)(1-p_t(x)))
6.     D_t(x) = p_t(x)(1 - p_t(x))
7.     h_t = L(D, y_t, D_t)
8.     H_t(x) = H_{t-1}(x) + (1/2)h_t(x)
9. end
Output: H(x) = sign(_{t=1}^T h_t(x))
```

10. LPBoost - Enfoque de Programación Lineal

Formulación como problema de optimización:

$$\min_{\alpha_j, \xi_i} \sum_{j=1}^{T} \alpha_j + C \sum_{i=1}^{m} \xi_i$$

sujeto a: $y_i \sum_{j=1}^{T} H_{i,j}\alpha_j + \xi_i \geq 1$, $\xi_i \geq 0$, $\alpha_j \geq 0$

10.1. Algoritmo LPBoost

```
Input: Dataset D; Algoritmo base ; Parámetro ; Rondas T.
Process:
1. w_{1,i} = 1/m
2. _1 = 0
3. for t = 1,...,T:
4.     h_t = (D, w)
5.     if _{i=1}^m w_{t,i}y_i h_t(x_i)  _t then break
6.     H_{i,t} = h_t(x_i)
7.     (w_{t+1}, _{t+1}) = arg min_{w,}  sujeto a:
        _{i=1}^m w_i y_i h_j(x_i)   (jt)
        _{i=1}^m w_i = 1
        w_i  [0, 1/(m)] (i=1,...,m)
8. end
9. resolver  de la solución dual (w_{T+1}, _{T+1})
Output: H(x) = sign(_{t=1}^T _t h_t(x))
```

11. Extensión Multiclase

11.1. AdaBoost.M1

Extensión directa para múltiples clases, pero requiere que cada aprendiz tenga error menor a 1/2.

11.2. SAMME

Mejora sobre AdaBoost.M1:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) + \ln(|\mathcal{Y}|-1)$$

12. Tolerancia al Ruido

12.1. MadaBoost

Modificación que limita el crecimiento de pesos:

$$D_{t+1}(x) = \frac{D_1(x)}{Z'_t} \times \min\left\{1, \prod_{i=1}^{t} e^{-\alpha_i h_i(x) f(x)}\right\}$$

12.2. BrownBoost

Algoritmo tolerante al ruido:

$$\ell_{\text{bmp}} = \mathbb{E}_{x\sim D}\left[1 - \operatorname{erf}\left(\frac{f(x)H_{t-1}(x) + f(x)h_t(x) + c - t}{\sqrt{c}}\right)\right]$$

12.3. RobustBoost

Mejora de BrownBoost:

$$\ell_{\text{oup}} = \mathbb{E}_{x\sim D}\left[1 - \operatorname{erf}\left(\frac{\hat{m}(H_{t-1}(x) + h_t(x)) - \mu(t/c)}{\sigma(t/c)}\right)\right]$$

13. Conclusiones

AdaBoost es un algoritmo poderoso que:

- Convierte aprendices débiles en fuertes
- Minimiza la pérdida exponencial
- Funciona bien en práctica aunque teóricamente podría sobreajustar
- Tiene extensiones para múltiples clases y datos ruidosos
- Se puede interpretar desde perspectivas de márgenes y estadística

La clave está en que ajusta iterativamente los pesos de las instancias, dando más importancia a las que son difíciles de clasificar, y combina inteligentemente los aprendices débiles ponderándolos según su performance.