2
Boosting at Start

Boosting at Start (BAS) (Mil09a, Mil09b, Mil09c) é uma nova abordagem de aprendizado de máquina baseado em Boosting. O BAS generaliza o algoritmo AdaBoost, permitindo a utilização de qualquer distribuição inicial para os exemplos. Neste capítulo, o algoritmo BAS é apresentado, bem como a sua prova de corretude.

2.1
Boosting

A construção de classificadores de boa qualidade, através do uso de aprendizado de máquina, é uma tarefa um tanto difícil. Todavia, é geralmente fácil criar regras simples, que possuem um desempenho melhor que uma simples resposta aleatória.

Um exemplo claro de regra simples para o problema de previsão de chuva é o seguinte:

Se a umidade relativa do ar for alta, então a previsão é de tempo chuvoso.

Tal regra, com certeza, não tem 100% de acurácia. Mais do que isso, seu desempenho é apenas ligeiramente superior ao da previsão aleatória. A estratégia de Boosting (Fre90, Sch90) baseia-se na combinação de diversas regras desse tipo para gerar um classificador. O classificador resultante apresenta um desempenho melhor do que uma regra simples e é menos custoso de construir do que uma regra complexa.

O Boosting funciona em iterações. A cada iteração, um **algoritmo-base** é chamado para gerar um classificador simples, utilizando uma diferente versão do conjunto de dados de treinamento. As diferentes versões do conjunto de treinamento são obtidas através da variação do peso associado a cada um dos exemplos. Assim, temos diferentes versões ponderadas do conjunto de dados. Após um número determinado de iterações, o Boosting combina os diversos classificadores parciais, gerando um classificador único, que, possivelmente, possui um desempenho melhor do que o do melhor classificador parcial.

Todos os algoritmos derivados do Boosting adotam esse mesmo esquema geral. A diferença está em dois pontos importantes do algoritmo: a forma de

atualização da ponderação do conjunto de dados em cada iteração e a forma de combinação dos classificadores parciais.

Uma forma simples de atualização dos pesos é aumentar os pesos dos exemplos classificados incorretamente pelos classificadores parciais. O objetivo é gerar classificadores parciais cada vez mais especializados nos exemplos *difíceis*.

Uma forma simples de combinação dos classificadores parciais é por votação. Neste caso, para cada classificador é atribuído um voto e a classificação final é decidida através de uma votação pela maioria.

As duas primeiras implementações de Boosting foram desenvolvidas por Robert Schapire, *Aprendizado Boosting* (Sch90) e por Yoav Freund, *Boosting pela Maioria* (Fre90). Entretanto, tais implementações não eram muito eficientes e sofriam de problemas de adaptabilidade em relação ao algoritmo-base utilizado, conforme ilustrado por experimentos conduzidos em tarefas de reconhecimento de caracteres (Dru93).

2.2 AdaBoost

O AdaBoost (Fre95a), ou Boosting Adaptativo, é um meta-algoritmo de aprendizado de máquina baseado em Boosting que pode ser utilizado em conjunto com qualquer algoritmo de aprendizado de máquina de forma a melhorar o seu desempenho em um determinado conjunto de dados. Os diversos classificadores, que compõem o seu comitê, são gerados sequencialmente. Cada classificador adicional é construído favorecendo os exemplos do conjunto de treinamento incorretamente classificados pelos classificadores anteriores.

O AdaBoost chama um algoritmo-base em várias iterações t, onde $t \in [1..T]$. Em cada iteração t, a distribuição de pesos do conjunto de treinamento é atualizada para utilização pelo algoritmo-base. A atualização é realizada de forma a, relativamente, aumentar os pesos dos exemplos incorretamente classificados em confronto com os pesos dos exemplos corretamente classificados.

Vamos representar o conjunto de dados de treinamento por $\{(x_i, y_i)\}_{i=1}^{n}$ onde $x_i \in X$ e $y_i \in \{-1, +1\}\}$. A distribuição de pesos do conjunto de dados, na iteração t, é denotada por D_t. Inicialmente, todos os pesos são iguais. A cada rodada, os pesos dos exemplos classificados incorretamente são aumentados. Dada a distribuição D_t, o algoritmo-base deve então achar um classificador apropriado h_t, que minimize o erro ponderado de treinamento ϵ_t, definido por

$$\epsilon_t = \sum_{\forall i | y_i \neq h_t(x_i)} D_t(i)$$

Caso o algoritmo-base não suporte a utilização de pesos diretamente, um esquema de reamostragem do conjunto deve ser utilizado.

Para cada classificador h_t, o AdaBoost determina um parâmetro α_t, que mede a importância do classificador h_t no comitê final formado.

Pode ser provado (Fre95a) que, nestas condições, o valor ótimo de α_t que minimiza o erro de treinamento é dado por

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

A distribuição de pesos D_t pode, então, ser atualizada utilizando a seguinte fórmula

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

Nesta fórmula, Z_t é uma constante de normalização, que transforma D_{t+1} em uma distribuição de probabilidade, e é denotada por

$$Z_t = \sum_{i=1}^{n} D_t(i) e^{-\alpha_t y_i h_t(x_i)}$$

Finalmente, o classificador combinado final H, formado pelos classificadores parciais gerados nas T iterações do algoritmo, cada um com poder de voto igual a α_t, é determinado pela seguinte expressão

$$H(x) = sinal \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

onde $sinal(x)$ é a função tal que

$$sinal(x) = \begin{cases} -1 & \text{se } x \text{ menor que zero,} \\ +1 & \text{caso contrário.} \end{cases}$$

Na verdade, nem sempre é possível gerar um comitê com T classificadores. Caso o parâmetro α_t encontrado em cada iteração for igual a zero, ou muito pequeno, o classificador h_t deve ser rejeitado, pois tal classificador não acrescenta em nada para o comitê final, seu poder de voto é nulo, ou praticamente nulo, e também não permite a geração de um novo classificador diferente dele. Nesse caso, o algoritmo é interrompido e o classificador final é formado pelos T', $T' < T$, classificadores já gerados.

Cabe ressaltar aqui também, que o valor de α_t encontrado para cada iteração minimiza o erro no conjunto de treinamento. Nenhuma suposição pode ser feita a respeito do comportamento do algoritmo em um conjunto de teste independente. Entretanto, caso os conjuntos de treinamento e teste possuam características similares, espera-se o mesmo comportamento também no conjunto de teste.

O pseudo-código do AdaBoost é mostrado no Algoritmo 2.1.

Algoritmo 2.1 AdaBoost.

Entrada: conjunto de exemplos: $Tr = \{(x_i, y_i)\}_{i=1}^{n}$ onde $x_i \in X$ e $y_i \in \{-1, +1\}\}$
algoritmo-base: Ab
número de iterações: T
para $i = 1$ **até** n **faça**
  $D_1(i) = 1/n$ // Inicialize a distribuição inicial
fim para
para $t = 1$ **até** T **faça**
  $h_t = Ab(Tr, D_t)$ // Treine o algoritmo-base utilizando a distribuição D_t e obtenha o classificador $h_t : X \rightarrow \{-1, +1\}$
  $\epsilon_t = \sum_{\forall i | y_i \neq h_t(x_i)} D_t(i)$ // Calcule a taxa de erro ponderado do classificador h_t
  $\alpha_t = \frac{1}{2} ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$ // Calcule o poder de voto do classificador
  $Z_t = \sum_{i=1}^{n} D_t(i) e^{-\alpha_t y_i h_t(x_i)}$ // Calcule a constante de normalização
  para $i = 1$ **até** n **faça**
    $D_{t+1}(i) = D_t(i) e^{-\alpha_t y_i h_t(x_i)} / Z_t$ // Atualize a distribuição de exemplos
  fim para
fim para
Saída: o classificador final
  $H(x) = sinal \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$

2.2.1 Exemplo do algoritmo AdaBoost

Podemos ilustrar a execução do algoritmo AdaBoost por meio do exemplo da Figura 2.1. Neste exemplo, desejamos aplicar o AdaBoost utilizando um algoritmo-base que encontra, para cada iteração, o segmento de reta perpendicular a um dos dois eixos que separa as duas classes e comete o menor número de erros.

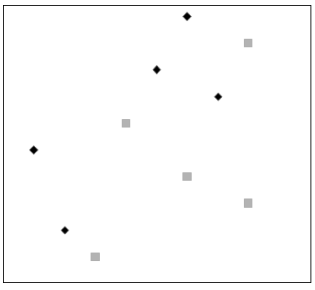

Figura 2.1: Dados do exemplo de utilização do AdaBoost.

Na Figura 2.2, mostramos os três classificadores-base gerados pelas iterações do algoritmo AdaBoost. Podemos perceber que cada classificador-base, isoladamente, classifica incorretamente três exemplos.

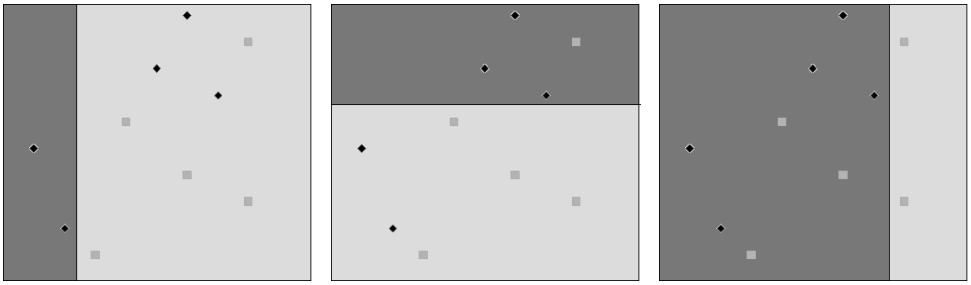

Figura 2.2: Iterações do exemplo de utilização do AdaBoost.

Na Figura 2.3, apresentamos o classificador final formado pelo esquema de combinação final do AdaBoost para os três classificadores mostrados na Figura 2.2. Este classificador, diferentemente de seus três componentes, não classifica nenhum exemplo incorretamente.

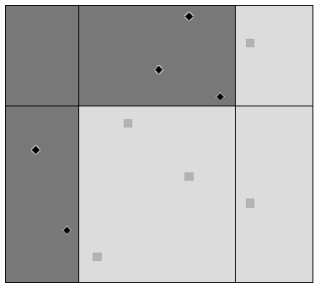

Figura 2.3: Classificador final do exemplo de utilização do AdaBoost.

2.2.2 AdaBoost.M1

Até agora, todas as nossa suposições se valeram do fato do AdaBoost gerar classificadores para problemas binários. Entretanto, existe uma maneira simples de se transformar o AdaBoost em um meta-algoritmo de classificação multi-classe. Tal algoritmo se chama AdaBoost.M1 (Fre95a).

Nesse caso, o conjunto Y possui tamanho k, e o algoritmo-base deve realizar um mapeamento $h_t : X \rightarrow \{1, ..., k\}$. A principal diferença é a hipótese final H que, dada uma instância x, tem como saída o valor de Y que maximiza a soma dos valores α_t que predizem uma classe.

Sendo assim, o classificador combinado final $H(x)$ pode ser expresso por

$$H(x) = \underset{y \in Y}{argmax} \left(\sum_{t=1}^{T} \alpha_t [h_t(x) = y] \right) \tag{2-1}$$

Um efeito colateral importante é que não podemos mais aceitar classificadores parciais cujo erro de treinamento ponderado é maior que $\frac{1}{2}$. No algoritmo

AdaBoost original, o classificador parcial h_t seria automaticamente substituído pelo seu complemento $1 - h_t$ que possui erro de treinamento ponderado menor que $\frac{1}{2}$. No AdaBoost.M1, caso tal fato ocorra, esse classificador se torna inútil para o classificador combinado final e o algoritmo deve ser interrompido usando apenas os classificadores parciais já encontrados.

2.2.3 Sensibilidade do AdaBoost à distribuição inicial

Consideremos agora o novo conjunto de dados da Figura 2.4 com o objetivo de novamente ilustrar a execução do algoritmo AdaBoost.

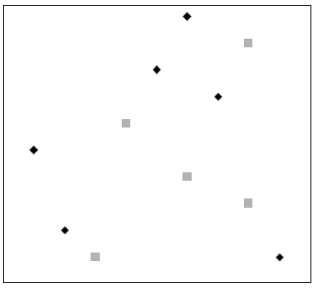

Figura 2.4: Novo conjunto de exemplos para execução do AdaBoost.

Na Figura 2.5, mostramos os três primeiros classificadores-base gerados pelo AdaBoost utilizando uma distribuição inicial uniforme. Para este conjunto, os classificadores-base não possuem todos o mesmo desempenho.

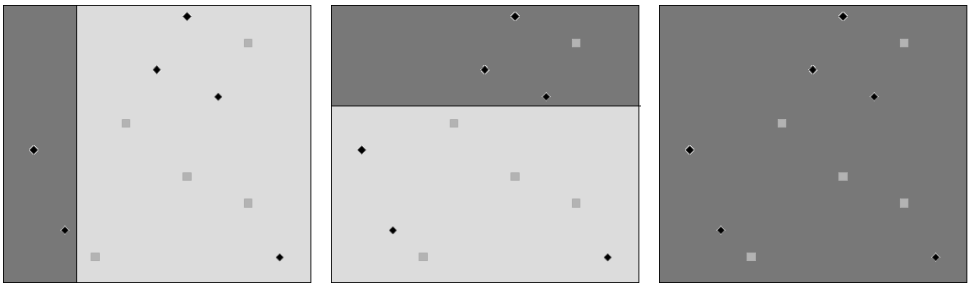

Figura 2.5: Iterações do AdaBoost para o novo conjunto.

Na Figura 2.6, apresentamos o classificador final formado pela combinação dos três classificadores gerados. Este classificador combinado possui uma taxa de erro de 18%.

Para este mesmo conjunto, mostramos agora a aplicação de uma variante do AdaBoost que permite a utilização de uma distribuição inicial arbitrária. Os pesos iniciais atribuídos são ilustrados na Figura 2.7 pelo tamanho de cada exemplo.

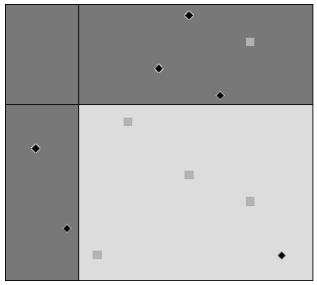

Figura 2.6: Classificador AdaBoost final para o conjunto modificado.

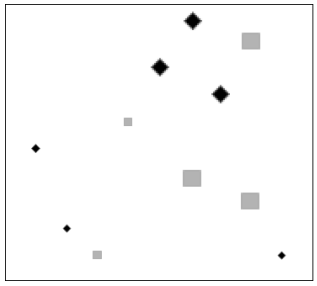

Figura 2.7: Distribuição inicial de pesos para o novo conjunto utilizado.

Na Figura 2.8, mostramos os três primeiros classificadores-base gerados por tal algoritmo que utiliza tal distribuição arbitrária. Diferentemente quando da utilização de uma distribuição inicial uniforme, todos classificadores-base possuem o mesmo desempenho, classificando incorretamente quatro exemplos cada.

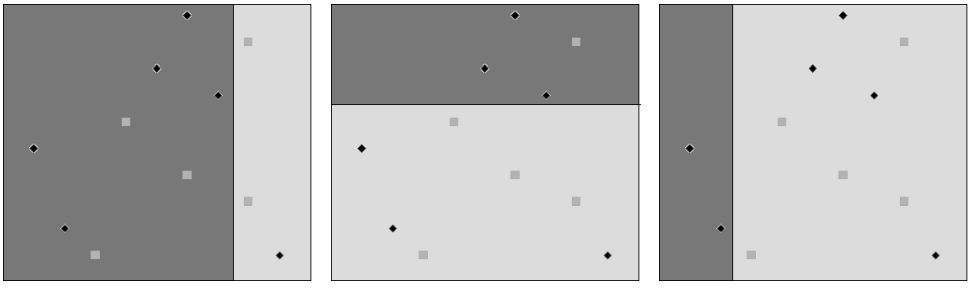

Figura 2.8: Iterações do algoritmo para o novo conjunto utilizando uma distribuição arbitrária.

Na Figura 2.9, apresentamos o classificador BAS final formado pela combinação dos três classificadores gerados. Este classificador combinado possui uma taxa de erro de 9%, menor que a do classificador final gerado utilizando a distribuição uniforme.

Além disso, utilizando uma distribuição uniforme, são necessárias quatorze iterações de forma a classificar todos os exemplos corretamente. Através

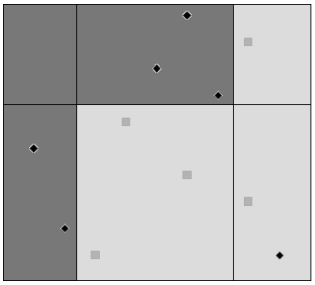

Figura 2.9: Classificador final gerado para o novo conjunto.

da distribuição arbitrária mostrada, por outro lado, são necessárias apenas nove iterações para gerar um classificador final com taxa de erro zero.

2.3 Boosting com uma distribuição arbitrária

Em seu trabalho inicial (Fre95a), Freund e Schapire vislumbraram a possibilidade de utilizar o AdaBoost com uma distribuição inicial diferente da uniforme. Seu objetivo, entretanto, era o de representar o conjunto de dados por meio apenas dos exemplos diferentes e da quantidade de vezes que eles apareceriam no conjunto de dados original. Com o objetivo de derivar as fórmulas de atualização dos pesos do AdaBoost, eles assumiram que a distribuição inicial dos exemplos do conjunto de dados é uniforme, visto que tais exemplos são amostrados de um conjunto maior infinito. Tal fato é exemplificado, por exemplo, na prova do limite do erro para o algoritmo AdaBoost utilizado no Teorema 6 da referência Freund & Schapire (Fre95a). Tal esquema foi chamado de *Boosting por Amostragem*. Toda a prova de corretude do algoritmo baseou-se nesta premissa.

Neste trabalho, introduziremos o Boosting at Start (BAS). Nosso objetivo aqui é permitir uma estratégia de Boosting que utilize uma distribuição inicial para os exemplos totalmente arbitrária, seja ela a original dos exemplos ou não, possibilitando dar uma maior diversidade aos classificadores parciais. Na Seção 2.4, apresentamos uma derivação formal do algoritmo. Tal derivação não assume nenhuma premissa sobre a distribuição inicial utilizada para os exemplos.

Suponhamos então que temos um conjunto de exemplos $\{(x_i, y_i)\}_{i=1}^{n}$ onde $x_i \in X$ e $y_i \in \{-1, +1\}\}$ e uma distribuição arbitrária para (x, y). Essa distribuição inicial é representada por uma função de pesos p que mapeia os exemplos (x, y) em $\Re$.

Agora, considere a possibilidade de reduzirmos o erro de treinamento nesse conjunto, trocando a distribuição D_1 original do algoritmo AdaBoost.

Podemos exprimir tal distribuição como sendo

$$D_1(i) \;=\; Kp(i), \qquad\qquad 0 \leq D_1(i) \leq 1$$

onde $i = 1, ..., n$ e K é uma constante de normalização definida por

$$K \;=\; \frac{1}{\sum_{i=1}^{n} p(i)}$$

Agora, podemos introduzir o *BAS*, uma variante do AdaBoost. No Algoritmo 2.2, mostramos o pseudo-código do algoritmo BAS.

A inovação é que o *BAS* aceita uma distribuição genérica para D_1. Dada essa modificação, um valor diferente para o α_t é necessário de forma a garantir que a taxa de erro no conjunto de treinamento do classificador combinado H seja diminuída.

Utilizando uma estratégia gulosa, mostramos na seção 2.4 que com o valor de α_t dado por

$$\alpha_t = \frac{1}{2} ln \left(\frac{\sum_{i \in C_t} D_t(i)/p(i)}{\sum_{i \in E_t} D_t(i)/p(i)} \right), \tag{2-2}$$

onde $C_t = \{i | h_t(x_i) = y_i\}$, $E_t = \{i | h_t(x_i) \neq y_i\}$, diminuímos o erro de treinamento do classificador final H.

É interessante notar que, quando D_1 é *uniforme*, temos o mesmo valor do algoritmo AdaBoost original, que é

$$\begin{aligned} \alpha_t &= \frac{1}{2} ln \left(\frac{\sum_{i \in C_t} D_t(i)}{\sum_{i \in E_t} D_t(i)} \right) \\ &= \frac{1}{2} ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \end{aligned}$$

Nesse caso, o BAS se reduz ao AdaBoost. Finalmente, observamos que a Equação (2-2) é geral, pois ela não realiza nenhuma suposição extra quanto à distribuição inicial D_1.

Alterações similares podem ser feitas em outros algoritmos baseados em Boosting, como por exemplo, TotalBoost (Fre96), LPBoost (Dem02) e SoftBoost (War07), de maneira a permitir o uso de qualquer distribuição inicial. Chamamos tal alteração no algoritmo original de *"basificar"* o algoritmo.

Algoritmo 2.2 BAS.

Entrada: conjunto de exemplos: $Tr = \{(x_i, y_i)\}_{i=1}^{n}$ onde $x_i \in X$ e $y_i \in \{-1, +1\}\}$
algoritmo-base: Ab
função de peso para os exemplos: p
número de iterações: T
$K = 1/\sum_{i=1}^{n} p(i)$ // Calcule a constante de normalização para os pesos iniciais
para $i = 1$ **até** n **faça**
  $D_1(i) = K.p(i)$ // Inicialize a distribuição inicial
fim para
para $t = 1$ **até** T **faça**
  $h_t = Ab(Tr, D_t)$ // Treine o algoritmo-base utilizando a distribuição D_t e obtenha o classificador $h_t : X \rightarrow \{-1, +1\}$
  $C_t = \{i|h_t(i) = y_i\}$ // Determine o conjunto de exemplos corretamente classificados
  $E_t = \{i|h_t(i) \neq y_i\}$ // Determine o conjunto de exemplos incorretamente classificados
  $\alpha_t = \frac{1}{2} ln \left(\frac{\sum_{i \in C_t} D_t(i)/p(i)}{\sum_{i \in E_t} D_t(i)/p(i)} \right)$ // Calcule o poder de voto do classificador
  $Z_t = \sum_{i=1}^{n} D_t(i) e^{-\alpha_t y_i h_t(x_i)}$ // Calcule a constante de normalização
  para $i = 1$ **até** n **faça**
    $D_{t+1}(i) = D_t(i) e^{-\alpha_t y_i h_t(x_i)}/Z_t$ // Atualize a distribuição de exemplos
  fim para
fim para
Saída: o classificador final
    $H(x) = sinal \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$

2.4 Corretude do Algoritmo BAS

Agora, vamos provar a corretude do cálculo do parâmetro α_t, dadas as alterações provocadas pela inserção de uma distribuição arbitrária inicial para os exemplos.

Seja o classificador combinado final H definido por

$$H(x) = sinal\,(f(x))$$

onde a função $f(x)$ é definida por

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

Consequentemente,

$$f(x) = H(x)|f(x)|.$$

O erro de treinamento τ para um preditor binário H é normalmente definido por

$$\tau = \frac{|E|}{n}$$

onde E é o conjunto dos exemplos incorretamente classificados, definido por

$$E = \{i|H(x_i) \neq y_i\}$$

Primeiro, observe que

$$\begin{aligned}
\sum_{i=1}^{n} e^{-y_i f(x_i)} &= \sum_{i=1}^{n} e^{-y_i H(x_i)|f(x_i)|} \\
&= \sum_{i \in E} e^{y_i^2 |f(x_i)|} + \sum_{i \notin E} e^{-y_i^2 |f(x_i)|} \\
&= \sum_{i \in E} e^{|f(x_i)|} + \sum_{i \notin E} e^{-|f(x_i)|} \\
&\geq \sum_{i \in E} 1 + \sum_{i \notin E} 0 = |E| = n\tau
\end{aligned}$$

Por esse motivo,

$$\tau \leq \frac{1}{n} \sum_{i=1}^{n} e^{-y_i f(x_i)}. \tag{2-3}$$

Por outro lado,

$$\begin{aligned}
\sum_{i=1}^{n} e^{-y_i f(x_i)} &= \sum_{i=1}^{n} \prod_{t=1}^{T} e^{-y_i \alpha_t h_t(x_i)} \\
&= \sum_{i=1}^{n} \prod_{t=1}^{T} Z_t \frac{D_{t+1}(i)}{D_t(i)} \\
&= \left(\prod_{t=1}^{T} Z_t \right) \sum_{i=1}^{n} \frac{D_{T+1}(i)}{D_1(i)}
\end{aligned}$$

Entretanto,

$$\frac{1}{n} \sum_{i=1}^{n} e^{-y_i f(x_i)} = \frac{\sum_{i=1}^{n} e^{-y_i f(x_i)}}{\sum_{i=1}^{n} D_1(i)/D_1(i)}$$

Logo,

$$\begin{aligned}
\frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(x_i)} &= \left(\prod_{t=1}^{T} Z_t\right).\frac{\sum_{i=1}^{n} D_{T+1}(i)/D_1(i)}{\sum_{i=1}^{n} D_1(i)/D_1(i)} \\
&= \left(\prod_{t=1}^{T} Z_t\right).\left(\prod_{t=1}^{T}\frac{\sum_{i=1}^{n} D_{t+1}(i)/D_1(i)}{\sum_{i=1}^{n} D_t(i)/D_1(i)}\right) \\
&= \prod_{t=1}^{T}\frac{\sum_{i=1}^{n} Z_t D_{t+1}(i)/D_1(i)}{\sum_{i=1}^{n} D_t(i)/D_1(i)} \\
&= \prod_{t=1}^{T}\left(\frac{e^{-\alpha_t}\sum_{i\in C_t} D_t(i)/D_1(i)}{\sum_{i=1}^{n} D_t(i)/D_1(i)}\right. \\
&\quad \left. + \frac{e^{\alpha_t}\sum_{i\in E_t} D_t(i)/D_1(i)}{\sum_{i=1}^{n} D_t(i)/D_1(i)}\right)
\end{aligned}$$

onde $C_t = \{i|h_t(i) = y_i\}$, $E_t = \{i|h_t(i) \neq y_i\}$. Tal equação pode ser reescrita como

$$\frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(x_i)} = \prod_{t=1}^{T} R_t \tag{2-4}$$

sendo R_t definido por

$$R_t = e^{-\alpha_t}\frac{\sum_{i\in C_t}^{n} D_t(i)/D_1(i)}{\sum_{i=1}^{n} D_t(i)/D_1(i)} + e^{\alpha_t}\frac{\sum_{i\in E_t}^{n} D_t(i)/D_1(i)}{\sum_{i=1}^{n} D_t(i)/D_1(i)} \tag{2-5}$$

Combinando as equações (2-3) e (2-4), chegamos a

$$\tau \leq \prod_{t=1}^{T} R_t. \tag{2-6}$$

E se definirmos a variável A_t por

$$A_t = \frac{\sum_{i\in E_t} D_t(i)/p(i)}{\sum_{i=1}^{n} D_t(i)/p(i)} \leq 1$$

R_t pode ser re-escrito como

$$R_t = e^{-\alpha_t}(1 - A_t) + e^{\alpha_t}(A_t)$$

Em cada iteração de Boosting t, escolhemos gulosamente α_t de forma a minimizar R_t, obtendo

$$\frac{\partial R_t}{\partial \alpha_t} = -e^{-\alpha_t}(1 - A_t) + e^{\alpha_t} A_t = 0$$
$$\alpha_t = \frac{1}{2} ln \left(\frac{1 - A_t}{A_t} \right) \tag{2-7}$$

E para esse valor,

$$\begin{aligned} R_t &= \sqrt{\frac{A_t}{1 - A_t}}(1 - A_t) + \sqrt{\frac{1 - A_t}{A_t}}(A_t) \\ &= 2\sqrt{A_t(1 - A_t)} \leq 1 \end{aligned}$$

Sempre que A_t é menor que $1/2$, R_t reduz o erro de treinamento definido por τ. Nesse caso, h_t é incluído na função f e $\alpha_t \neq 0$ pode ser determinado por

$$\alpha_t = \frac{1}{2} ln \left(\frac{\sum_{i \in C_t} D_t(i)/p(i)}{\sum_{i \in E_t} D_t(i)/p(i)} \right) \tag{2-8}$$