2
Boosting

2.1.
Problema de Classificação

De acordo com Ponti & Mascarenhas (2004), a idéia de capacitar uma máquina para classificar padrões começou a evoluir em meados do século XX e, desde esta época, muitas aplicações surgiram para incentivar as pesquisas: reconhecimento de caracteres, identificação de impressões digitais, análise de terrenos em imagens de satélite, entre outras. Aprendizado de Máquina é a área da computação que trata destes problemas.

Para realizar qualquer tipo de classificação é necessário encontrar características inerentes a cada classe que possam diferenciar um objeto do outro. Por exemplo, no caso de uma aplicação de reconhecimento facial, poderíamos definir as seguintes características: posição do nariz, formato da face e cor dos olhos.

Este trabalho lida com classificadores cujo aprendizado é supervisionado, ou seja, utilizamos exemplos conhecidos de cada classe para gerar um preditor que, posteriormente, rotulará objetos desconhecidos. Este processo de geração do preditor através de exemplos é denominado treino.

2.2. Validação Cruzada

A classificação pode ser entendida, de maneira geral, pela partição do espaço de atributos em um número finito de regiões, fazendo com que objetos de uma mesma classe recaiam sobre a mesma região. Uma vez terminado o treino, é preciso verificar a qualidade do preditor através do teste. Chamamos de amostras os objetos conhecidos que fazem parte do teste.

Intuitivamente, pode-se pensar que quanto maior o número de exemplos no treinamento, melhor é a performance do preditor. No entanto, precisamos aprender o problema e não decorá-lo. Aprender é ser capaz de generalizar e, por isto, a quantidade de exemplos deve ser minimamente suficiente para tal. Já na etapa de teste, quanto maior for a quantidade de amostras, melhor será a precisão na avaliação da eficiência do preditor.

Em virtude da dificuldade, enfrentada na maioria dos problemas de classificação, em conseguir um numeroso conjunto de dados, torna-se necessário trabalhar com técnicas para contornar escassez de dados. Uma delas, denominada validação cruzada (*cross-validation*) e formalizada por Stone (1974) consiste em dividir o conjunto de dados, aleatoriamente, em *n* subconjuntos disjuntos de mesmo tamanho. Um destes subconjuntos é reservado para o teste e os restantes para o treino. Tal procedimento é repetido *n* vezes, utilizando, em cada uma delas, um subconjunto diferente para teste e os restantes para treino. Ao final, calculamos, através da eq. 1, a média entre as *n* classificações distintas.

$$E = \frac{1}{n}\sum_{i=1}^{n} m(i) \qquad (1)$$

A função m(i) representa a métrica utilizada na validação cruzada.

A validação cruzada, ilustrada na Figura 7, é amplamente utilizada na avaliação de classificadores.

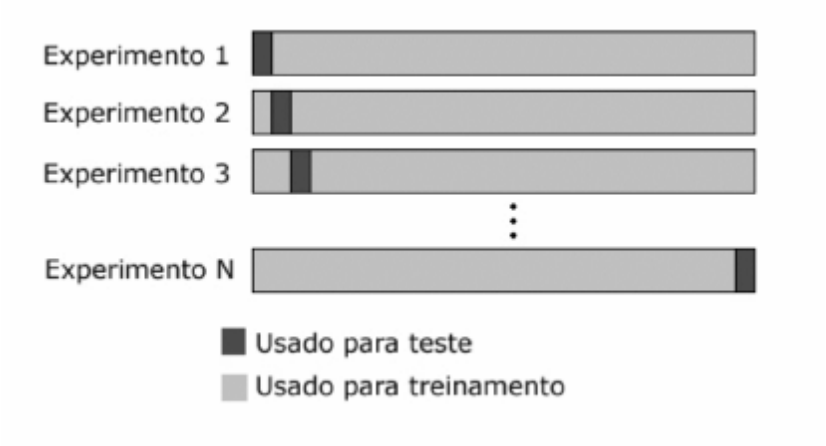

Figura 7: Exemplo de validação cruzada.

Porém, a validação cruzada aplicada nos nossos experimentos de anúncios (seções 4.4, 4.5 e 4.6) difere sutilmente da supracitada. A seleção do conjunto de teste é sempre aleatória nas *n* iterações (Figura 8). O resultado final será obtido através da eq. 1. Esta técnica é conhecida como *repeated holdout* (Kohavi, 1995).

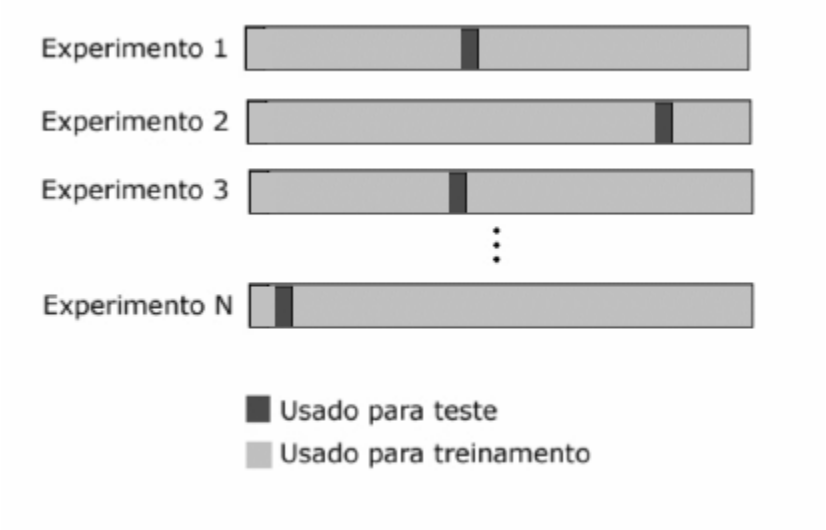

Figura 8: Exemplo de *repeated holdout*.

2.3. AdaBoost

Segundo Vidal et al. (2006), a principal idéia do *Boosting* é efetuar repetidas execuções de um classificador "base" (*base learner* ou *weak learner*) produzindo um comitê de preditores e combinar os resultados obtidos, gerando um único preditor eficiente. A essência da técnica consiste em modificar, a cada execução, os pesos dos exemplos no conjunto de treino. Tal procedimento possibilita que o próximo preditor gerado atue na falha do anterior. Os algoritmos de *Boosting* vêm produzindo bons resultados quando aplicados em modelos de Filtragem Colaborativa (DeCoste, 2006; Wu, 2007; Cai & Hofmann, 2003).

Na década de 90, Freund & Schapire (1996) propuseram um algoritmo denominado AdaBoost (***Ada****ptative* ***Boost****ing*). Originalmente, o objetivo do AdaBoost é otimizar uma classificação binária e, para isto, invoca diversas vezes um algoritmo "base" de classificação. Nos referiremos a essas invocações por "rodadas" daqui por diante. A cada rodada é utilizada uma distribuição de pesos diferente, representando a relevância dos exemplos e, ao seu final, o algoritmo calcula um coeficiente α. Este coeficiente é proporcional à confiança do preditor gerado naquela rodada.

A distribuição inicial de pesos é uniforme e, no decorrer das rodadas, o AdaBoost aumenta o peso de cada exemplo *i*, caso o preditor erre ou diminui, caso acerte. O erro total da rodada é definido pela eq. 2.

$$\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i] = \sum_{i:h_t(x_i)\neq y_i} D_t(i) \qquad (2)$$

Note que este erro total é o somatório, ponderado pelos pesos, dos exemplos classificados incorretamente. O coeficiente α_t é calculado em função de tal erro, conforme mostra a eq. 3.

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \tag{3}$$

Ao final de cada rodada, os pesos são atualizados conforme a eq. 4, ou seja, de acordo com o valor de α_t. A função h_t representa a predição do exemplo x_i:

$$D'_{t+1} = D_t(i) \times e^{-\alpha_t \times h_t(x_i)} \tag{4}$$

A distribuição de pesos precisa então ser normalizada após a atualização, garantindo que seus valores estejam entre 0 e 1. Na eq. 5, Z_t é o fator de normalização.

$$D_{t+1} = \frac{D'_{t+1}}{Z_t} \tag{5}$$

O resultado final, uma combinação linear de todos os coeficientes obtidos é dada pela eq. 6.

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t \times h_t(x)\right) \tag{6}$$

Após unir os conceitos acima apresentados, exibimos o algoritmo de Freund & Schapire (1996) na Figura 9.

Dado: $S = \{(x_1, y_1), \ldots, (x_m, y_m); x_i \in X, y_i \in \{-1, +1\}\}$

Algoritmo *Adaboost*:

Inicialize $D_1(i) := \frac{1}{m} \quad \forall \ (x_i, y_i) \in S$

For $t = 1, \ldots T$, faça

Treine o classificador base usando a distribuição D_t

Obtenha a hipótese fraca $h_t: X \rightarrow \{-1, +1\}$

Calcule α_t: $\alpha_t = \frac{1}{2} ln \frac{1 - e_t}{e_t}$, onde e_t é a taxa de erro do classificador h_t.

Atualize a distribuição:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} = \begin{cases} e^{-\alpha_t}, \text{ se } h_t(x_i) = y_i \\ e^{\alpha_t}, \text{ se } h_t(x_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Onde Z_t é o fator de normalização

End For

Saída: Hipótese Final: $H(x) = sign\left(\sum_{1..T} \alpha_t * h_t(x)\right)$

Figura 9: Algoritmo AdaBoost.

A única restrição do AdaBoost é que o algoritmo "base" de classificação não acerte, exatamente, 50% dos exemplos. Note que, se o erro for 0.5, α_t será 0. Se α_t for 0, os pesos não são atualizados, pois serão multiplicados por 1 no passo seguinte. Logo, o algoritmo não progredirá.

A seguir, exemplificamos uma execução do AdaBoost empregado a um classificador linear. Os preditores gerados cortam o plano na horizontal ou na vertical, separando-o em uma região positiva e outra negativa. A Figura 10 apresenta a disposição inicial dos exemplos.

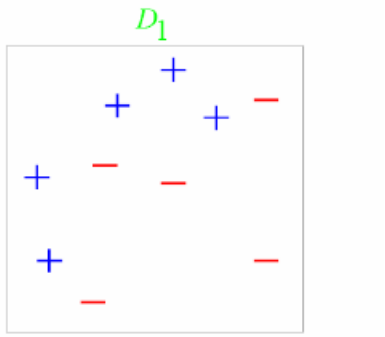

Figura 10: Distribuição dos exemplos na classe “+” ou na classe “-“.

Os exemplos classificados incorretamente estão “ampliados” (Figuras 11, 12 e 13) indicando que serão tratados com maior importância nas próximas rodadas. Na figura 14, exibimos a combinação linear resultante do comitê de três classificadores.

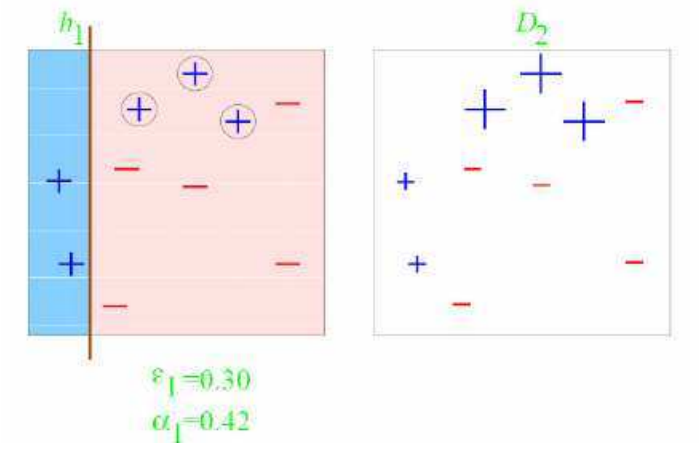

Figura 11: Executando a primeira rodada do AdaBoost.

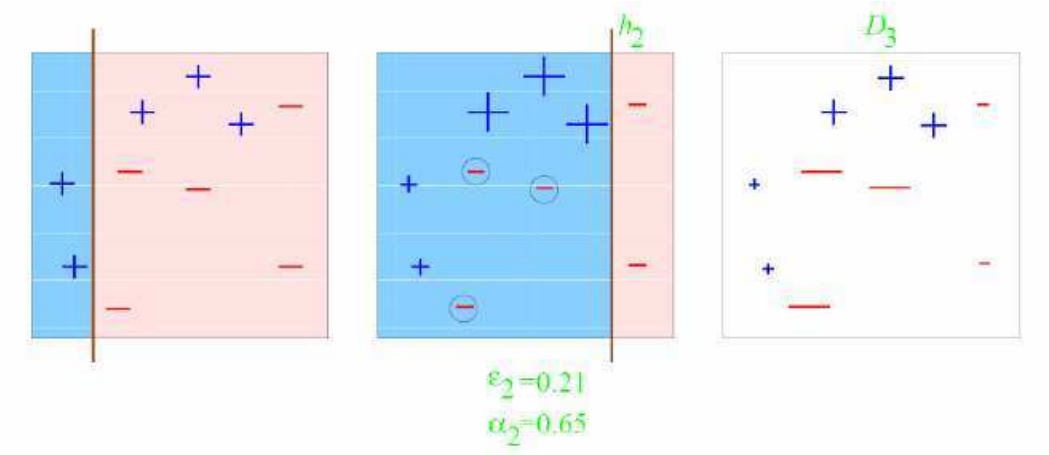

Figura 12: Executando a segunda rodada do AdaBoost.

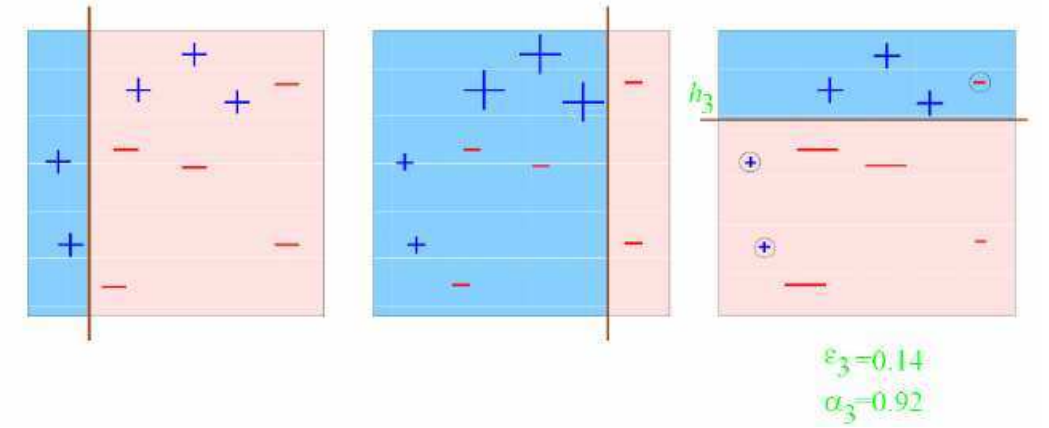

Figura 13: Executando a terceira e última rodada do AdaBoost.

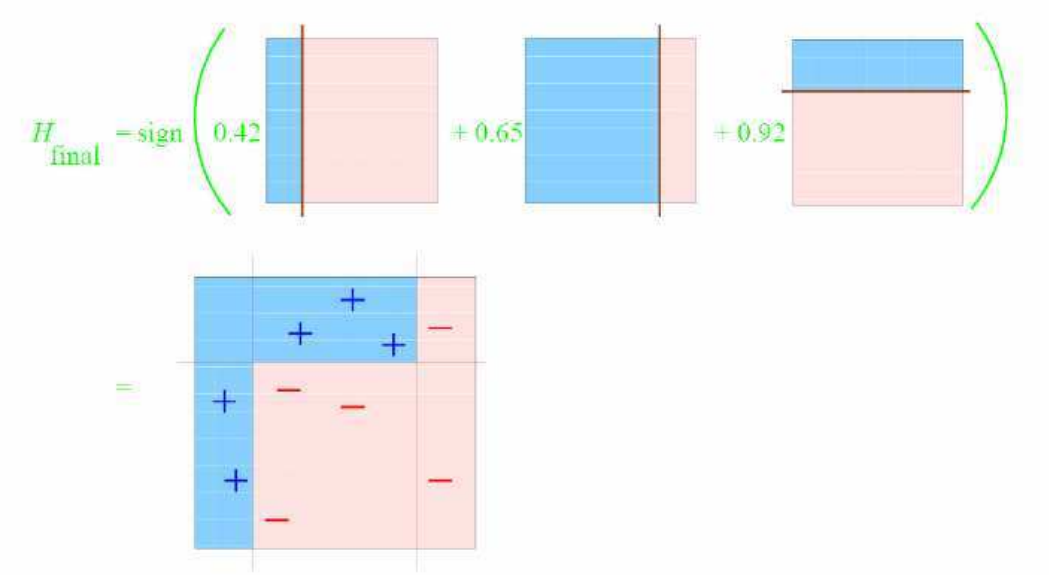

Figura 14: Combinando os resultados.