Boosting

Victor Freguglia Souza
RA: 137784
and
Leonardo Uchoa Pedreira
RA: 156231

2 de Dezembro de 2018

Resumo

The text of your abstract. 200 or fewer words.

Keywords: Quando for a versão final., Tirar esse campo na .tex;

1 Introdução

Boosting é o nome dado a um tipo de algoritmo que, assim como outros métodos em Machine Learning como Bagging e Florestas Aleatórias, busca combinar um grande número de preditores com baixo poder de predição (isto é, um pouco mais eficientes do que a escolha ao acaso) para compor um bom preditor. O conceito é fundamentado nas ideias apresentadas em Kearns (1988) e Schapire (1990).

Diferentemente dos outros métodos citados, onde os preditores fracos que serão combinados são criados de maneira independente (e aleatória devido ao processo de bootstrap), no método de Boosting os preditores fracos são criados de maneira a melhorar o desempenho em regiões com altas taxas de erro. Isto é, se pensarmos que cada preditor tem um "voto" na decisão final, o método nos fornece um comitê em que aqueles que tem grande convicção têm mais poder na decisão. Estes de grande convicção, sabem muito sobre uma parte do espaço amostral (não sei se amostral é a melhor palavra).

O algoritmo AdaBoost (de *Adaptative Boosting*), apresentado pela primeira vez em Freund & Schapire (1997), é um dos exemplos mais clássicos de algoritmo de boosting para o problema de classificação binária. Nele, a cada passo m, um novo classificador G_m é ajustado com base em uma versão ponderada do conjunto de dados original, na qual o peso de cada observação depende do desempenho do classificador anterior: pontos classificados de maneira errada recebem peso maior, e assim, têm uma chance maior de serem corrigidos pelos classificadores ajustados na próxima iteração. O Algoritmo 1 apresenta a descrição completa do AdaBoost. Note que os classificadores G_m a serem usados não precisam ser de nenhum tipo específico e, portanto, se comporta como um parâmetro a ser escolhido por um processo de regulagem (tuning) do problema. Apesar disso, o mais comum, pela grande flexibilidade, é a árvore de classificação e regressão (CART).

Embora o AdaBoost seja um bom ponto de início para entender o conceito de Boosting, ele foi projetado para resolver problemas de classificação binária, e por isso, não apresenta resultados tão bons quando diretamente adaptado a problemas de classificação múltipla e regressão, então diferentes algoritmos são necessários para esses casos. Hastie et al. (2009) propõe uma modificação do AdaBoost para o caso de classificação múltipla, por exemplo.

A principal diferença entre os diferentes algoritmos de Boosting propostos está na ma-

início

$(y_i, x_i), i = 1, \ldots, N$; $y_i \in \{-1, 1\}$; $x_i \in \mathbb{R}^p$;

Inicie todos pesos $w_i = 1/N$;

para $m = 1, \ldots, M$ **faça**

1. Ajuste um classificador $G_m(x)$;
2. Calcular

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i};$$

3. Calcular $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$;
4. Atualizar $w_i \leftarrow w_i \exp\left(\alpha_m I(y_i \neq G_m(x_i))\right), i = 1, \ldots, N$;

fim

Defina o classificador final como

$$G(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m I(G_m(x) = k)).$$

fim

Algoritmo 1: Algoritmo AdaBoost apresentado em Friedman et al. (2001). Aqui, as classes do problema de classificação são representadas pelos valores -1 e 1.

neira com que os a ponderação dos dados é feita em cada passo. Nesse trabalho, por apresentar uma forma mais geral, será considerado o algoritmo Gradient Boosting, que pode lidar com todos os tipos de problemas de predição com a devida escolha da função perda a ser minimizada. A teoria que o fundamenta e sua descrição detalhada são apresentadas na Seção 2. O método é então aplicado ao conjunto de dados MNIST, descrito na Seção 3 e os resultados são mostrados na Seção 4.

2 Metodologia

2.1 Contexto e Visão Geral

Em problemas de regressão, escolher o melhor preditor significa estimar uma função f^* que minimiza o risco, definido como o valor esperado da função de perda $L(\cdot, \cdot)$, ou seja,

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{Y|x}\big[L(Y, f(x))|x\big]. \tag{1}$$

Note que o risco para um preditor específico f é desconhecido, uma vez que não assumimos nenhuma distribuição, o que impossibilita o cálculo do valor esperado da função perda para esse preditor. Uma boa estratégia para criação da função de predição $f \in \mathcal{F}$ consiste em escolher preditores de forma a minimizar o risco empírico. Isto é,

$$f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{2}$$

Uma forma de resolver o problema 2 é via utilização de algorítmos numéricos. Gradiente descendente (citar watkins), por exemplo, é uma estratégia habitualmente utilizada, que levaria ao algorítmo

$$f_m = f_{m-1} - \rho_k \sum_{i=1}^{N} \nabla_f L(y_i, f(x_i)) \tag{3}$$

Entretanto (Friedman (2001)) cita que simplesmente utilizar isto teria efeitos catastróficos no modelo preditivo. Primeiro porquê só são levados em consideração as observações que temos (i.e, não existe nenhuma intenção de extrapolar poder preditivo, como por exemplo a separação entre teste e treinamento ou validação cruzada faria), segundo que não é levado em consideração à relação entre as covariáveis. Para contornar este problema, o autor sugere sequencialmente (de forma aditiva) ajustar uma quantidade m de árvores de decisão aos pseudo-resíduos, obtidos de predições subsequentes. Assim, o problema de otimização 2 se torna

$$f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{4}$$

sujeito à que f seja árvore,

e que oferece como preditor $f_{boost}(x) = \sum_{m=1}^{M} \text{Árvore}_m$.

2.2 Árvores de decisão

Árvore de decisão é uma técnica que busca criar partições disjuntas R_j, $j = 1, \ldots, J$ em que γ_j é uma constante associada à cada partição terminal (Friedman et al. (2001)). Neste caso, se uma observação x pertence à região j, a predição para ela será γ_j. Formalmente, tal árvore pode ser escrita como

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{5}$$

onde $\Theta = \{R_j, \gamma_j\}$, $j = 1, \ldots, J$. Bem, se nosso objetivo é usar várias árvores em 4, precisamos de uma maneira de estimar Θ. Ou seja, na ótica da Teoria de Decisão, para a m-ésima árvore (ou seja, m-ésimo passo) queremos

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x; \Theta)), \tag{6}$$

pois como citado anteriormente (2.1) o ajuste é feito de maneira aditiva e nos resíduos. Para isto existem métodos que fornecem as regiões R_j, como o Particionamento Recursivo (Friedman et al. (2001)). Mas agora que conhecemos as regiões R_j da árvore m, considere que nos encontramos em uma partição terminal j dela, isto é, $I(x \in R_j) = 1$, $\forall x \in R_j$. Neste caso, o que nos falta é

$$\hat{\gamma_{jm}} = \arg\min_{\gamma_{jm}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma_{jm}), \tag{7}$$

que dependerá da função de perda escolhida. Friedman et al. (2001) fornece uma tabela de soluções analíticas para γ, de acordo com algumas perdas. Assim, conseguimos obter Θ_m, $\forall m = 1, \ldots, M$.

2.3 Otimização por Gradiente

Gradiente Descendente (ou seu irmão, Mais Íngrime Descida) é um método de otimização numérica que busca obter o mínimo de uma função. Sua proposta é começar em um ponto

inicial e incrementar a função avaliada naquele na direção oposta ao gradiente, em uma certa "velocidade" ρ_k, como em 3.

Entretanto, ao considerar que temos um problema de minimização com restrição o algorítmo torna-se (Friedman et al. (2001))

$$f_m = f_{m-1} - \rho_k \sum_{i=1}^{N} \left[\nabla_f L(y_i, f(x_i))\right]_{f(x_i)=f_{m-1}(x_i)} \tag{8}$$

onde

$$\rho_k = \arg\min_{\rho} L(f_{m-1}(x_i) - \rho\left[\nabla_f L(y_i, f(x_i))\right]_{f(x_i)=f_{m-1}(x_i)}) \tag{9}$$

2.4 Gradient Boosting

A direção fornecida da solução pelo método do Gradiente e sua conexão com os resíduos fornecem a intuição e a engrenagem por detrás de Boosting. Boosting é baseado em modelos aditivos, onde sequencialmente se ajustam modelos lineares generalizados nos resíduos. Ou seja, no passo m, temos o problema

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta_m b(x_i; \gamma_m)) \tag{10}$$

onde b é uma função de base. Se, por exemplo, usamos a perda quadrática o problema torna-se

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N} (r_{m-1}(x_i) - \beta_m b(x_i; \gamma_m))^2, \tag{11}$$

em que $r_{m-1}(x_i) = y_i - f_{m-1}(x_i)$. Ao compararmos a equação acima ao problema de regressão linear simples, $r_{m-1}(x_i)$ toma o papel de y_i e $\beta_m b(x_i; \gamma_m)$, o papel de βx_i. Portanto, a analogia leva à conclusão de se ajustar uma função de base aos resíduos. Boosting funciona de forma similar, onde γ_m torna-se Θ_m.

Se olharmos a equação equação 6, a predição da árvore $T(x_i; \Theta)$ é justamente o argumento que fornece o mínimo. Este é o mesmo papel que o valor negativo do gradiente desempenha em 2, o que os tornam similares, em certo sentido. Além disso, também existe

a semelhança entre 7 e 9 (onde a diferença é que, no caso das árvores, a busca pela solução ótima é restrita ao nó terminal).

Para perceber onde tudo se encaixa, vamos voltar ao exemplo da perda quadrática. Na equação 8,

$$\left[\nabla_f L(y_i, f(x_i))\right]_{f(x_i)=f_{m-1}(x_i)} = -2(y_i - f_{m-1}(x_i)) := -g_{im}.$$

Isto fornece

$$\begin{aligned}
\hat{\Theta}_m &= \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x;\Theta)) \\
&= \arg\min_{\Theta_m} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - T(x;\Theta))^2 \\
&= \arg\min_{\Theta_m} \sum_{i=1}^{N} (-gim - T(x;\Theta))^2.
\end{aligned}$$

De acordo com a analogia anterior para o caso de modelos aditivos, g_{im} aqui tem uma forte conexão com os resíduos (para este exemplo, ele de fato é). Na verdade, ele é chamado de *pseudo-resíduo*, pois para problemas de classificação, os "resíduos" são, na verdade, a margem de classificação (Friedman et al. (2001) cita como exemplo a perda exponencial e sua interpretação para o algorítmo AdaBoost). Esta é o argumento chave que está por detrás da idéia do Boosting, a conexão entre os pseudo-resíduos e a direção do gradiente.

Se juntarmos as idéias e formularmos um algorítmo, temos o boosting por gradiente (ou

gradient boosting) como feito em Friedman (2001). Isto fornece o algorítmo a seguir.

início

$(y_i, \mathbf{x}_i), i = 1, \ldots, N$;

Inicie com o preditor constante $H_0 = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$;

para $m = 1, \ldots, M$ **faça**

1. Calcular
$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f}\right]_{G=f_{m-1}(x_i)};$$
2. Ajustar uma nova árvore f_m ao conjunto de dados $(r_{im}, \mathbf{x}_i)$ com J regiões terminais $R_{jm}, j = 1, \ldots, J$
3. Para $j = 1, \ldots, J$, obtenha
$$\hat{\gamma_{jm}} = \arg\min_{\gamma_{jm}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma_{jm});$$
4. Atualize $f_m(x_i) = f_{m-1}(x_i) + \sum_{j=1}^{J} \gamma_{jm} I(x_i \in R_{jm})$;

fim

O preditor final é então $f_{boost}(x) = f_M(x)$.

fim

Algoritmo 2: Algoritmo Gradient Boosting apresentado em @boosting2001.

Importante: Boosting surgiu para problemas de classificação binária e foi adaptado para regressão e classificação de várias categorias. Para classificação de K classes, para $m = 1, \ldots, M$, ajuste K árvores de classificação binária e use uma função perda apropriada, como a softmax (Friedman (2001)), que é a mais habitual para este tipo de problema.

2.5 PCA Whitening

Talvez isso não seja necessário pois habitualmente as arvores nao precisam desse tipo de tratamento. Svm e redes habitualmente precisam pq usam demais combinacoes lineares.

Mas essa secao pode ser util pra fazer comparacao com os outros metodos em que isso serve e melhorar a secao de discussao.

3 Dados

Conjunto de dados MNIST.

- Mostrar alguns números

4 Aplicação

- Ajuste
- Comparar com Random Forest, nnet, pca-whitened

5 Discussão

- Não sei o que entra em discussão/conclusão?

Proposta:

Discutir (pq em aplicação não há discussão crítica) os resultados da comparação do gbm com os métodos anteriores em questão de velocidade, precisão, flexibilidade. Considerar as variantes xgboost, lightgbm

Discutir efeito do pca no gbm e em outros métodos que precisem disso, como redes e svm.

Usar outros aspectos considerados no arquivo "kuhn_cheatsheet" pra comparar?

6 Conclusão

Fazer por último e ver o que aprendemos do trabalho.

Referências

Freund, 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), 119–139.

Friedman, J. H. (2001), 'Greedy function approximation: a gradient boosting machine', *Annals of statistics* pp. 1189–1232.

Friedman, J., Hastie, T. & Tibshirani, R. (2001), *The elements of statistical learning*, Vol. 1, Springer series in statistics New York, NY, USA:.

Hastie, T., Rosset, S., Zhu, J. & Zou, H. (2009), 'Multi-class adaboost', *Statistics and its Interface* **2**(3), 349–360.

Kearns, M. (1988), 'Thoughts on hypothesis boosting', *Unpublished manuscript* **45**, 105.

Schapire, R. E. (1990), 'The strenght of weak learnability', *Machine Learning* **5**, 197–227.