Fundamentos de Mineração de Dados e Ciência de Dados

Algoritmos de classificação para fluxo de dados

05/11/2025

1 Desafios do Aprendizado em Fluxos de Dados

Algoritmos tradicionais de aprendizado de máquina, como o CART, foram projetados para cenários *batch* (em lote), onde todo o conjunto de dados está disponível na memória principal e pode ser processado múltiplas vezes. Em fluxos de dados, esses algoritmos são inviáveis devido a três limitações principais:

1. **Limitação de Memória**: É impossível armazenar todos os exemplos do fluxo, que é potencialmente infinito.
2. **Limitação de Tempo**: Os dados chegam em alta velocidade, exigindo que cada exemplo seja processado em tempo (quase) constante, geralmente em uma única passada.
3. **Modelos a Qualquer Momento (*Any-Time*)**: O algoritmo deve ser capaz de fornecer um modelo de classificação viável a qualquer instante, mesmo que não tenha terminado de processar todo o fluxo.

O algoritmo VFDT (*Very Fast Decision Tree*) foi desenvolvido para superar esses desafios, utilizando a abordagem *top-down* dos algoritmos tradicionais de árvores de decisão, porém de forma incremental e estatisticamente robusta.

Outra abordagem é a construção de *ensembles* (comitês) de classificadores, cujo desempenho é monitorado e, caso algum dos classificadores perca desempenho, é substituído por outro mais recente. O KUE (*Kappa Updated Ensemble*) é um exemplo desta abordagem.

2 O Algoritmo VFDT (Very Fast Decision Tree)

2.1 Ideia Geral e o Limite de Hoeffding

A ideia central do VFDT é simples: em vez de usar todos os dados para decidir como expandir a árvore (como fazem os algoritmos em *batch*), acumula exemplos

em uma folha até ter **evidência estatística suficiente** para escolher o melhor atributo para fazer uma divisão (*split*).

O componente crucial é o **Limite de Hoeffding**. Seja $H(\cdot)$ uma função de avaliação de atributos (como Ganho de Informação). Se observarmos n exemplos em uma folha, o Limite de Hoeffding nos garante que, com probabilidade $1-\delta$, a diferença entre a avaliação média do melhor atributo na amostra ($\overline{H}(X_a)$) e sua avaliação esperada na distribuição verdadeira é de no máximo ϵ, onde:

$$\epsilon = \sqrt{\frac{R^2 \ln(2/\delta)}{2n}},$$

em que R é o valor máximo possível da função $H(\cdot)$. No caso do ganho de informação baseado na entropia, $R = \log_2(k)$, onde k é o número de classes (veremos essa demonstração abaixo).

Seja X_a o melhor atributo na amostra e X_b o segundo melhor. Se a diferença $\overline{\Delta H} = \overline{H}(X_a) - \overline{H}(X_b)$ for maior que ϵ, podemos ter confiança estatística de que X_a é realmente o melhor atributo e transformar a folha em um nó de decisão que se divide em X_a. Mas esta decisão só será tomada quando n for suficientemente grande (note que ϵ é inversamente proporcional a n).

Em outras palavras, um nó

Calculando R: Vamos usar o exemplo do Ganho de Informação baseado na entropia, visto na aula 5: suponha que estejamos em um nó folha t, e queremos decidir se iremos ou não dividi-lo.

- A probabilidade (desconhecida) de uma instância incidente sobre um nó t da árvore pertencer à classe k é estimada pela frequência relativa desta classe no conjunto de instâncias incidentes no nó t:

  $$\hat{\pi}_k = \frac{n_k}{n},$$

  onde n é o número de instâncias incidentes no nó t e n_k é o número de instâncias da classe k incidentes em t.

- O critério mais simples e usual de rotulagem é baseado na classe majoritária:

  $$k^* = \arg\max_{k=1\ldots K} \hat{\pi}_k.$$

- A entropia das classes no nó é calculada como:

  $$\mathcal{E}(t) = -\sum_{k=1}^{K} \hat{\pi}_k \cdot \log_2[\hat{\pi}_k]$$

- Neste caso, o maior ganho de informação possível ocorrerá quando o nó t tiver uma distribuição homogênea de classes e tiver uma divisão cujos nós filhos $t_1, t_2, \ldots$ sejam "puros" (cada nó filho só contém instâncias de uma classe):

$$\begin{aligned}
\Delta\mathcal{E} &= \mathcal{E}(t) - \left[\frac{N_{t_1}}{N_t}\mathcal{E}(t_1) + \frac{N_{t_2}}{N_t}\mathcal{E}(t_2) + \ldots\right] \\
&= \mathcal{E}(t) - 0 \\
&= -\sum_{k=1}^{K} 1/k \cdot \log_2[1/k] \;=\; -1/k \sum_{k=1}^{K} \log_2[1/k] \\
&= -1/k \cdot k \cdot \log_2[1/k] \;=\; -\log_2[1/k] \;=\; \log_2[k]
\end{aligned}$$

2.2 Síntese dos Passos do Algoritmo

1. **Inicialização**: A árvore começa como uma única folha (a raiz).
2. **Trajeto**: Para cada novo exemplo, percorre-se a árvore da raiz até uma folha, seguindo os testes de divisão nos nós internos.
3. **Atualização**: Na folha alcançada, atualizam-se as **estatísticas suficientes** para todos os atributos. Essas estatísticas são contagens de valores de atributos por classe, necessárias para calcular funções como o Ganho de Informação.
4. **Avaliação**: Após um número mínimo de exemplos (N_{min}) chegar à folha, avalia-se a possibilidade de dividi-la.
5. **Decisão de Divisão**: Calcula-se ϵ e a diferença $\overline{\Delta H}$ entre os dois melhores atributos. Se $\overline{\Delta H} > \epsilon$, a folha é substituída por um nó de decisão baseado no melhor atributo. Novas folhas vazias são criadas como seus filhos.
6. **Desempate**: Se ϵ se tornar muito pequeno ($\epsilon < \tau$) mas $\overline{\Delta H}$ ainda for menor que ϵ, força-se a divisão pelo melhor atribo para evitar indecisão.

2.3 Pseudocódigo e Explicação

Parâmetros e Estatísticas:

- N_{min}: Controla o balanço entre velocidade e acurácia. Um valor maior economiza cálculos, mas pode atrasar as divisões.
- $1 - \delta$: A probabilidade de uma divisão estar incorreta. Um δ menor (ou seja, $1 - \delta$ maior) exige mais exemplos para decidir.

Algorithm 1 VFDT: A Árvore de Hoeffding

Entrada: S: Fluxo de exemplos; X: Conjunto de atributos; Y: Classes; $H(.)$: Função de divisão; N_{min}: Número mínimo de exemplos para particionar uma folha; δ: nível de significância; τ: erro aceitável quando o Limite de Hoeffding não for atingido.

Saída: HT: Uma Árvore de Decisão

$HT \leftarrow$ Folha Vazia (Raiz)
para cada exemplo $(x, y_k) \in S$ **faça**
    Percorra HT da raiz até uma folha l
    Atualize as estatísticas suficientes em l
    se número de exemplos em $l > N_{min}$ **então**
        Calcule $H_l(X_i)$ para todos os atributos
        Seja X_a o atributo com maior H_l
        Seja X_b o atributo com segundo maior H_l
        Calcule $\epsilon = \sqrt{\frac{R^2 \ln(2/\delta)}{2n}}$
        se $(H(X_a) - H(X_b)) > \epsilon$ **então**
            Substitua l por um teste de divisão em X_a
            Adicione uma nova folha vazia para cada ramo da divisão
        senão se $\epsilon < \tau$ **então** ▷ Resolve empates
            Substitua l por um teste de divisão em X_a
            Adicione uma nova folha vazia para cada ramo da divisão
        fim se
    fim se
fim para

- τ: Constante de desempate para quando dois atributos são quase igualmente bons.

- **Estatísticas Suficientes**: Em cada folha, para cada atributo, armazenamos uma tabela de contagens $Count(Atributo, Valor, Classe)$.

  Para atributos categóricos, a atualização dessa tabela tem custo baixo: basta incrementar a tabela na respectiva posição.

  Para atributos contínuos, estruturas mais complexas (como árvores-B) são usadas para manter as contagens necessárias para calcular todos os pontos de corte possíveis.

2.4 Extensão para Atributos Contínuos: Análise Discriminante

Na maioria dos problemas reais de aprendizado de máquina, os atributos numéricos. Para esse tipo de atributo, a divisão de um nó é descrita por um teste do tipo $X_i \leq d$ (onde d é o ponto de corte).

Decidir o ponto de corte ideal para um atributo contínuo em algoritmos *batch* geralmente requer ordenar os valores, uma operação custosa ($O(n \log n)$). Em fluxos de dados, essa abordagem é proibitiva.

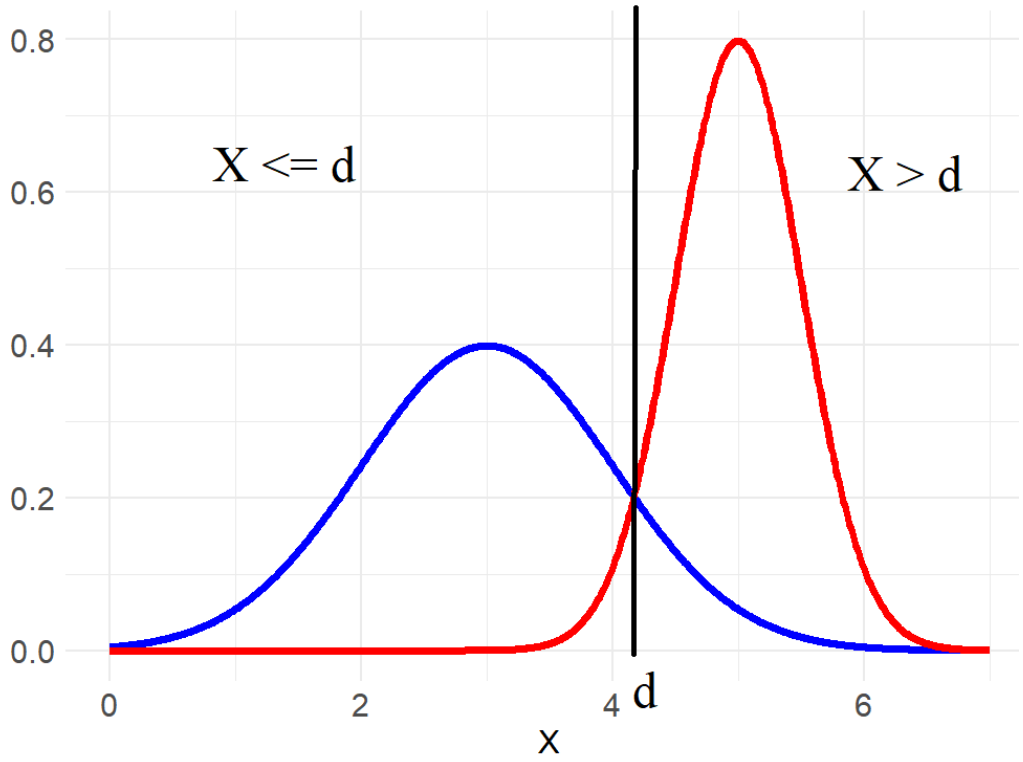
Um método eficiente, proposto por Gama et al. (2004), usa Análise Discriminante para encontrar um bom ponto de corte em problemas de classificação binária, sem necessidade de ordenação.

O método assume que os valores do atributo seguem uma distribuição normal para cada classe. Para um problema de duas classes (+ e −), o ponto de corte d ideal é a solução da equação:

$$P(+)N(\bar{x}_+, \sigma_+) = P(-)N(\bar{x}_-, \sigma_-)$$

onde $N(\bar{x}, \sigma)$ é a função de densidade da Normal, e $P(+)$, $P(-)$ são as probabilidades a priori das classes. A raiz d mais próxima das médias das classes é escolhida como candidata a ponto de corte.

A figura abaixo ilustra esta ideia.



Para que o método seja viável, a média ($\bar{x}$) e a variância (σ^2) de cada atributo por classe devem ser atualizadas incrementalmente a cada novo exemplo, sem relembrar todos os valores anteriores.

A atualização direta da variância pela fórmula $\sigma^2 = \frac{\sum x_i^2}{n} - \bar{x}^2$ é problemática numericamente, podendo levar a erros de precisão (cancelamento catastrófico) quando a variância é pequena comparada à média.

Uma abordagem numericamente estável, conhecida como **Algoritmo de Welford**, atualiza as estatísticas da seguinte forma:

Para uma sequência de valores $x_1, x_2, \ldots, x_n$:

- Inicialize: $n = 0$, $\bar{x}_0 = 0$, $M_2 = 0$
- Para cada novo valor x_{n+1}:

$$\begin{aligned} n &= n + 1 \\ \delta &= x_{n+1} - \bar{x}_n \\ \bar{x}_{n+1} &= \bar{x}_n + \delta/n \\ M_2 &= M_2 + \delta \cdot (x_{n+1} - \bar{x}_{n+1}) \end{aligned}$$

- A variância amostral é então $\sigma_n^2 = \frac{M_2}{n-1}$ (para $n > 1$).

Este método é computacionalmente eficiente ($O(1)$ por atualização) e numericamente estável, sendo ideal para fluxos de dados. No VFDT, manteríamos esses

contadores (n, $\bar{x}$, M_2) para cada atributo contínuo em cada folha, separadamente por classe.

3 Comentários Finais e Propriedades

O VFDT e suas extensões compartilham as propriedades desejáveis das árvores de decisão tradicionais, como **interpretabilidade** e **robustez** a transformações monotônicas dos atributos. Além disso, possuem propriedades específicas que os tornam ideais para fluxos de dados:

- **Estabilidade**: A decisão de divisão, baseada no Limite de Hoeffding, confere ao algoritmo uma baixa variância, gerando modelos similares para conjuntos de dados similares. Isso contrasta com a alta variância de algoritmos gulosos como o CART.
- **Classificador a Qualquer Momento**: Como as divisões são feitas de forma mais balanceada ao longo do tempo (sempre que há evidência estatística), a árvore é um classificador útil desde os primeiros instantes.
- **Convergência**: Foi demonstrado que a árvore gerada pelo VFDT é assintoticamente próxima daquela que seria gerada por um algoritmo *batch* que visse todos os dados.

Em resumo, o VFDT estabeleceu um marco para o aprendizado de árvores de decisão em fluxos de dados, combinando fundamentos estatísticos sólidos com eficiência computacional, e abrindo caminho para uma família de algoritmos adaptativos e escaláveis.

4 Kappa Updated Ensemble

4.1 Ideia Geral do Algoritmo

O *Kappa Updated Ensemble* (KUE) é um método de ensemble projetado especificamente para classificação em fluxos de dados que apresentam *concept drift*. Desenvolvido por Cano e Krawczyk (2020), o KUE tem como característica principal utilizar o coeficiente Kappa de Cohen como métrica para avaliar o desempenho dos classificadores base e orientar a atualização dinâmica do ensemble.

A ideia central do algoritmo é manter um conjunto de classificadores diversos, onde cada especialista é treinado em um subespaço diferente de características

(*feature subspace*). Periodicamente, novos classificadores são treinados e comparados com os piores especialistas do ensemble usando a estatística Kappa. Se um novo classificador demonstrar desempenho superior, ele substitui o pior classificador do ensemble, garantindo assim uma evolução contínua e adaptativa do modelo.

O KUE opera processando os dados em chunks (lotes) e é particularmente eficaz para problemas com distribuição balanceada de classes, embora não seja especificamente otimizado para cenários de desbalanceamento.

4.2 O Coeficiente Kappa de Cohen

O coeficiente Kappa de Cohen (κ) é uma métrica estatística que mede o grau de concordância entre as classes preditas por um classificador e as classes reais, ajustada pelo grau esperado de concordância por um "classificador aleatório".

Sua definição é dada por:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

onde:

- p_o: é a acurácia observada (proporção de classificações corretas)
- p_e: é a acurácia esperada ao acaso

Assim, a intuição fundamental do Kappa é quantificar quanto o classificador em construição performa melhor do que um classificador aleatório que apenas adivinha seguindo a distribuição das classes.

- $\kappa = 1$: concordância perfeita (o classificador é perfeito)
- $\kappa = 0$: o classificador performa exatamente como esperado por acaso
- $\kappa < 0$: o classificador é pior que o acaso (as classes preditas e as verdadeiras estão "invertidas")

No contexto do KUE, o Kappa é particularmente útil porque:

- É **insensível ao desbalanceamento de classes**, ao contrário da acurácia simples
- Fornece uma **avaliação mais robusta** do desempenho real do classificador

- Permite **comparar classificadores** de forma mais justa, mesmo quando operam em subespaços diferentes

A política de descartar votos de classificadores com $\kappa < 0$ no KUE garante que apenas modelos que performam melhor que o acaso influenciem a decisão final do ensemble.

4.3 Pseudocódigo e Descrição Detalhada

O Algoritmo 2 apresenta o pseudocódigo do Kappa Updated Ensemble, e pode ser dividido em quatro fases principais:

1. **Inicialização do Ensemble** (linhas 2-8):
   - Ao receber o primeiro chunk S_1, o ensemble é inicializado com k classificadores.
   - Para cada classificador γ_j:
     - É selecionado aleatoriamente um número $r \in [1, f]$ que define o tamanho do subespaço de características (linha 4).
     - O método EscSubEspCar seleciona aleatoriamente r características do conjunto total (linha 5).
     - O classificador γ_j é treinado no chunk S_1 filtrado apenas para as características selecionadas (linha 6).
     - A estatística Kappa do classificador é calculada e armazenada (linha 7).
2. **Atualização do Modelo do Ensemble** (linhas 10-13):
   - Para chunks subsequentes S_i $(i > 1)$, cada classificador existente é atualizado incrementalmente.
   - Cada classificador γ_j é treinado no novo chunk S_i filtrado pelo seu subespaço φ_j.
   - A estatística Kappa de cada classificador é recalculada com base no desempenho mais recente.
3. **Treinamento de Novos Componentes** (linhas 15-18):
   - Para cada um dos q novos componentes a serem treinados:
     - Um novo subespaço de características é selecionado aleatoriamente.
     - Um novo classificador γ' é treinado no chunk atual S_i usando este subespaço.

- O desempenho do novo classificador é avaliado através da estatística Kappa κ'.

4. **Substituição de Componentes Fracos** (linhas 19-24):

   - O método KappaMinimo identifica o classificador com o menor valor de Kappa no ensemble atual.
   - Se o novo classificador γ' tiver um Kappa superior ao do pior classificador do ensemble ($\kappa' > \kappa_{\min}$), ocorre a substituição.
   - O classificador mais fraco na posição w é substituído pelo novo classificador, juntamente com seu subespaço e estatística Kappa.

Algorithm 2 KUE: Kappa Updated Ensemble

Entrada: S: fluxo de dados, f: número de características, k: número de componentes do ensemble, q: número de novos componentes para treinar

Saída: ε: ensemble de k classificadores, φ: subespaço de características para cada componente, κ: estatística Kappa para cada componente

1: **para** cada $S_i \in \{S_1, \ldots, S_n\}$ **faça** ▷ Processa cada chunk do fluxo
2: **se** S_1 **então** ▷ Inicialização do ensemble
3: **para** $j \in \{1, \ldots, k\}$ **faça** ▷ Para cada classificador no ensemble
4: $r \leftarrow$ inteiro aleatório com probabilidade uniforme $[1, f]$
5: $\varphi_j \leftarrow$ EscSubEspCar(r)
6: $\gamma_j \leftarrow$ treinar novo classificador em FiltSubEspCar(φ_j, S_1)
7: $\kappa_j \leftarrow$ calcular Kappa de γ_j
8: **fim para**
9: **senão** ▷ Atualização do modelo do ensemble
10: **para** $j \in \{1, \ldots, k\}$ **faça** ▷ Atualiza cada classificador existente
11: $\gamma_j \leftarrow$ treinamento incremental de γ_j em FiltSubEspCar(φ_j, S_i)
12: $\kappa_j \leftarrow$ calcular Kappa de γ_j
13: **fim para**
14: **para** $m \in \{1, \ldots, q\}$ **faça** ▷ Treinar novos componentes
15: $r \leftarrow$ inteiro aleatório com probabilidade uniforme $[1, f]$
16: $\varphi' \leftarrow$ EscSubEspCar(r)
17: $\gamma' \leftarrow$ treinar novo classificador em FiltSubEspCar(φ', S_i)
18: $\kappa' \leftarrow$ calcular Kappa de γ'
19: $\kappa_{\min}, w \leftarrow$ KappaMinimo(ε)
20: **se** $\kappa' > \kappa_{\min}$ **então** ▷ Substituir o mais fraco $\gamma \in \varepsilon$
21: $\varphi_w \leftarrow \varphi'$
22: $\gamma_w \leftarrow \gamma'$
23: $\kappa_w \leftarrow \kappa'$
24: **fim se**
25: **fim para**
26: **fim se**
27: **fim para**

4.4 Mecanismo de Classificação

Para classificar novas instâncias, o KUE utiliza o método de *votação majoritária*. Classificadores com estatística Kappa abaixo de um limiar (p.ex. 0.5) têm seus votos descartados, evitando que modelos de baixo desempenho influenciem negativamente a decisão final do ensemble.

4.5 Comentários Finais

A diversidade e adaptabilidade do KUE é garantida através de:

- **Subespaços aleatórios**: Cada classificador opera em um subconjunto diferente de características (estratégia similar às *Random Forests*).
- **Substituição dinâmica**: A substituição contínua dos piores classificadores por novos especialistas treinados nos dados mais recentes.
- **Atualização incremental**: Todos os classificadores são atualizados com cada novo chunk de dados.

Esta abordagem confere ao algoritmo uma forte capacidade de adaptação a mudanças de conceito (*concept drift*) ao longo do fluxo de dados.

Referências

- Gama, J. (2010). Knowledge Discovery from Data Streams. CRC Press. Pré-print disponível em `http://www.liaad.up.pt/area/jgama/DataStreamsCRC.pdf`.
- Oliveira, D. A. (2023). BASWE: Balanced Accuracy-based Sliding Window Ensemble for classification in imbalanced data streams with concept drift. Dissertação de Mestrado, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo. doi:10.11606/D.100.2023.tde-01012024-172026. Disponível em `https://doi.org/10.11606/D.100.2023.tde-01012024-172026`.