Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Estatística
Programa de Especialização em *Data Science* e *Big Data*

Fabio Miecznikowski

Avaliação do custo computacional de algoritmos de detecção de desvio de conceito

Curitiba
2024

Fabio Miecznikowski

Avaliação do custo computacional de algoritmos de detecção de desvio de conceito

Monografia apresentada ao Programa de Especialização em *Data Science* e *Big Data* da Universidade Federal do Paraná como requisito parcial para a obtenção do grau de especialista.

Orientador: Prof. Paulo R. Lisboa de Almeida

Curitiba
2024

Avaliação do custo computacional de algoritmos de detecção de desvio de conceito

Evaluation of the Computational Cost of Concept Drift Detection Algorithms

Fabio Miecznikowski[1], Paulo R. Lisboa de Almeida[2]

[1]Aluno do programa de Especialização em Data Science & Big Data, fabiokwk@gmail.com
[2]Professor do Departamento de Informatica - UFPR, paulorla@ufpr.br

Este artigo analisa o desempenho de algoritmos de detecção de concept drift em fluxos de dados utilizando a árvore de Hoeffding e compara com a aplicação desse classificador sem nenhum gatilho. Foram aplicadas as técnicas ADWIN, KSWIN, DummyDriftDetector, PageHinkley, HDDM_W, HDDM_A, EDDM e a árvore de Hoeffding sem nenhum gatilho em oito conjuntos de dados reais: Forest Covertype, Electricity, Poker-Hand, Airlines, HTTP, KeyStroke, Phishing, ImageSegments. A avaliação focou em métricas de tempo de execução, uso da CPU, uso da memória e acurácia, com auxílio das seguintes bibliotecas Python: scikit-multiflow, psutil e time. Os resultados indicam que a aplicação da configuração padrão dos gatilhos nem sempre irá superar acurácia do classificador em questão, mas que é possível obter ganhos em acurácia e uso de memória(MB). Observou-se que é possível ter ganhos de acurácia, tanto ao custo de mais tempo de processamento como de menos tempo. Também, é possível obter ganhos de acurácia com menos consumo de memória.
Palavras-chave: Concept drift, Hoeffding Tree, ADWIN, KSWIN, DummyDriftDetector, PageHinkley, HDDM_W, HDDM_A, EDDM

This article analyzes the performance of concept drift detection algorithms in data streams using the Hoeffding's tree and compares it with the application of this classifier without any trigger. The techniques ADWIN, KSWIN, DummyDriftDetector, PageHinkley, HDDM_W, HDDM_A, EDDM, and the Hoeffding's tree without any trigger were applied to eight real datasets: Forest Covertype, Electricity, Poker-Hand, Airlines, HTTP, KeyStroke, Phishing, ImageSegments. The evaluation focused on metrics of execution time, CPU usage, memory usage, and accuracy, with the help of the following Python libraries: scikit-multiflow, psutil, and time. The results indicate that applying the default trigger configurations will not always surpass the accuracy of the classifier in question, but it is possible to achieve improvements in both accuracy and memory usage (MB). It was observed that accuracy gains can be achieved either at the cost of more processing time or with less time. Additionally, accuracy improvements can be obtained with lower memory consumption.
Keywords: Concept drift, Hoeffding Tree, ADWIN, KSWIN, DummyDriftDetector, PageHinkley, HDDM_W, HDDM_A, EDDM

1. Introdução

Nos últimos anos, o campo da ciência de dados tem enfrentado desafios significativos devido à natureza dinâmica e não estacionária dos fluxos de dados. Um desses desafios críticos é o concept drift, que ocorre quando a relação entre os dados de entrada e a variável alvo muda ao longo do tempo [1]. Esse fenômeno pode ser exacerbado em ambientes onde a distribuição dos dados muda gradualmente, mesmo que a distribuição das entradas permaneça constante[1].

O concept drift real refere-se especificamente às mudanças na distribuição condicional da saída, dado um conjunto de entradas, introduzindo a necessidade de métodos adaptativos para atualizar modelos preditivos em tempo real[1]. A detecção eficaz de concept drift torna-se crucial em cenários onde a estabilidade do modelo preditivo é fundamental para a precisão e confiabilidade das previsões[3].

Além disso, fluxos de dados são caracterizados como sequências ininterruptas de exemplos recebidos a uma taxa tão alta que cada exemplo só pode ser processado uma vez, tornando impraticável o armazenamento completo de todos os dados na memória[2]. Esta limitação implica em estratégias de processamento efi-

cientes para lidar com a alta taxa de chegada de dados sem comprometer a qualidade da análise[1].

Este artigo tem por objetivo analisar o comportamento de algoritmos de detecção de concept drift em fluxos de dados. Dessa forma, foi explorado o método de aprendizagem online Hoeffding Tree juntamente com as técnicas de identificação de desvio - ADWIN[6], KSWIN[6], DummyDriftDetector[7], PageHinkley[6], HDDM_W[7], HDDM_A[7] e EDDM[3] - em diversos conjuntos de dados reais, que já são conhecidos no contexto do aprendizado de máquinas: Forest Covertype, Electricity, Poker-Hand, Airlines, HTTP, KeyStroke, Phishing, ImageSegments. A análise dessas técnicas visa proporcionar conhecimento sobre custo computacional para saber como lidar eficazmente com mudanças nos padrões dos dados ao longo do tempo, mantendo a acurácia e relevância dos modelos preditivos.

2. Discussão

Esta seção revisa trabalhos que exploram o uso de gatilhos de detecção de deriva de conceito. A literatura existente destaca estratégias e metodologias para lidar com esse fenômeno em ambientes de aprendizado incremental.

O fenômeno de concept drift pode ser classificado em duas categorias principais: concept drift virtual e real. No concept drift virtual, há mudanças na distribuição dos dados de entrada sem afetar P(y|X)[2]. Aqui, X representa as variáveis independentes ou características dos dados de entrada, y é a variável dependente, e P(y|X) é a probabilidade de y dado X. No concept drift real ocorre mudança em p(y|X), independente se há mudança em p(X)[4].Do ponto de vista preditivo, apenas mudanças que afetam a decisão de previsão requerem adaptação.[1]

Em um estudo recente, foi demonstrado que um classificador cego alcançou alta precisão no benchmark Electricity utilizando a abordagem de teste e treinamento sequencial. Sugere-se que métodos que empregam o conjunto de dados Electricity devem ser comparados com um classificador cego para avaliação adequada[9].

Outro trabalho mostrou que um gatilho cego, disparando periodicamente, superou alguns gatilhos de ponta nos benchmarks Electricity e Forest covertype. Além disso, foi proposta uma análise de métricas computacionalmente eficientes em benchmarks do mundo real, como Electricity e Airlines[10]. Este estudo contribuiu para ampliar a compreensão das estratégias viáveis para lidar com a deriva de conceito na aprendizagem incremental.

É crucial considerar a dinâmica do cenário de aprendizagem, incluindo taxas de deriva, natureza dos dados (online ou em lotes) e recursos computacionais disponíveis ao selecionar um algoritmo adequado para ambientes não estacionários[2].

Sobre gatilhos de detecção de desvio de conceito, são algoritmos que se concentram em identificar mudanças na distribuição de dados, ao monitorar os resultados de um classificador[8].

3. Materiais e métodos

Para avaliar o desempenho dos algoritmos de detecção de mudança de conceito, em conjunto com o algoritmo de árvore de Hoeffding, foram selecionados oito conjuntos de dados conhecidos no contexto de aprendizado de máquina, compostos por dados reais: Forest Covertype, Electricity, Poker-Hand, Airlines, HTTP, KeyStroke, Phishing e ImageSegments[11][12].

Dataset	Instâncias	Atributos
Poker-hand	1,000,000	11
Forest Covertype	581,012	54
HTTP	567,498	3
Airlines	539,383	7
Electricity	45,312	5
KeyStroke	20,400	31
ImageSegments	2,310	18
Phishing	1,250	9

Tabela 1: Instâncias e atritubos para cada dataset

Foram implementados sete gatilhos de detecção de mudança de conceito: ADWIN (Adaptive Windowing), KSWIN (Kolmogorov-Smirnov Window), DummyDriftDetector, PageHinkley, HDDM_W, HDDM_ A e EDDM. O algoritmo de árvore de Hoeffding foi escolhido devido à sua capacidade de aprendizado online e de processar grandes volumes de dados em fluxo contínuo [13].

A implementação dos experimentos foi realizada utilizando Python com a biblioteca scikit-multiflow ,para o classificador, em conjunto com river, para aplicação dos gatilhos. Além disso, tanto a árvore de Hoeffding como os algoritmos de detecção de mudança foram aplicados com a configuração padrão, não houve nenhum ajuste no modelo.

O treinamento e a avaliação foram conduzidos utilizando o algoritmo árvore de Hoeffding como base para

cada script de detecção de mudança. A cada gatilho acionado, o modelo foi retreinado com as últimas 100 instâncias.

As métricas utilizadas para avaliação incluíram tempo de execução, uso da CPU, uso da memória e acurácia.

4. Resultados

Esta seção contém os resultados da avaliação da acurácia, tempo de execução e consumo de memória dos algoritmos de detecção de concept drifts utilizando os oito conjuntos de dados reais apresentados na Seção [1].

A Tabela2 apresenta os resultados médios dos diferentes algoritmos de detecção de mudança aplicados com o modelo Hoeffding Tree, analisando quatro métricas: acurácia média, tempo total médio, uso médio da CPU e memória usada média.

Algoritmo	Acurácia	Tempo(s)	CPU (%)	Memória(MB)
NONE	74.79	31.52	6.27	218.55
ADWIN	76.70	26.77	6.68	191.19
KSWIN	76.86	63.71	6.73	191.42
DUMMY	74.33	31.62	6.55	192.02
PAGE	75.94	31.67	6.68	192.52
$HDDM_W$	74.74	62.27	6.52	196.54
$HDDM_A$	74.68	65.69	6.50	198.65
EDDM	75.48	27.19	6.47	195.55

Tabela 2: Média dos resultados

A análise dos dados revelou que o método KSWIN obteve a maior acurácia média, alcançando 76,86%, seguido pelo ADWIN com 76,70%. Esses resultados sugerem que esses métodos são mais eficazes na adaptação às mudanças nos padrões dos dados. Em contrapartida, o método DummyDriftDetector apresentou a menor acurácia média, com 74,33%. A menor acurácia desse modelo já era esperada, visto que ele não é um detector de desvio e apenas serve de base para avaliar os demais modelos. Dessa forma, observa-se que a escolha do método pode impactar significativamente a precisão do modelo preditivo.

Em termos de tempo total de execução, o método ADWIN foi o mais rápido, com uma média de 26,77 segundos, seguido pelo EDDM com 27,19 segundos. O método KSWIN apresentou o maior tempo médio de execução, com 63,71 segundos. Portanto, ao considerar a implementação prática, o tempo de execução é um fator crítico para a eficiência operacional.

Quanto ao uso da CPU, o método KSWIN teve o maior consumo médio, com 6,73%, enquanto o EDDM apresentou o menor consumo médio, com 6,47%. A diferença no uso da CPU entre os métodos é relativamente pequena. Isso indica que, apesar das variações na acurácia e no tempo de execução, o impacto no uso da CPU é minimamente variado.

Em relação à memória utilizada, o método NONE (árvore de Hoeffding sem nenhum gatilho) apresentou o maior consumo médio, com 218,55 MB, o que era esperado, já que ele não implementa nenhuma técnica de detecção de drifts, resultando em menos operações de limpeza de memória. O método ADWIN foi o mais econômico em termos de memória, com uma média de 191,19 MB. Os outros métodos apresentaram valores intermediários, com diferenças não muito significativas. Possivelmente, o uso do Python pode ser responsável por uma parcela do consumo de memória. Assim, a utilização da memória é um aspecto importante a ser considerado ao selecionar o método apropriado para ambientes com restrições de recursos.

5. Conclusão

Os custos computacionais observados neste estudo podem embasar a escolha de algoritmos de detecção de concept drift em diferentes cenários. Em ambientes com alta taxa de chegada de dados e restrições de tempo, como sistemas de monitoramento em tempo real, algoritmos que oferecem menor tempo de execução, como o ADWIN, podem ser preferíveis.

Em contraste, para aplicações onde a precisão é fundamental e o volume de dados é grande, como em análises de dados históricos ou pesquisas de grandes conjuntos de dados, o KSWIN pode ser o mais adequado, caso o contexto tolere um tempo de processamento mais longo. Para essa opção, seria necessário ajustar o modelo a fim de obter melhor acurácia.

Além disso, para sistemas com restrições de memória, como dispositivos embarcados, algoritmos que equilibram o uso de memória e precisão, como o ADWIN, podem ser a melhor opção. Visto que, todos os gatilhos foram utilizados com sua configurações padrão, ainda é possível obter melhores resultados.

A escolha do algoritmo deve, portanto, considerar o custo-benfício entre acurácia, tempo de processamento e consumo de recursos, alinhando-se às necessidades específicas da aplicação.

A árvore de Hoeffding, sem estar em conjunto com algum gatilho, apresentou maior uso de memória em comparação com os demais algoritmos, nos quais havia o classificador e o gatilho operando em conjunto. Possivelmente, isso ocorre, devido ao fato de que esses retreinavam o modelo a cada gatilho com as últimas

100 instâncias e aquele mantinha o modelo desde a primeira. Além disso, parte do uso de memória pode ser atribuída ao uso do Python.

Em resumo, compreender os custos computacionais dos algoritmos de detecção de concept drift permite uma seleção mais informada e eficiente, adaptada às demandas e limitações de cada cenário de aplicação.

Pesquisas futuras poderiam se concentrar na otimização desses algoritmos e também na aplicação de um número maior de conjuntos de dados, assim como a utilização de mais gatilhos.

6. Agradecimentos

Desejo expressar minha sincera gratidão a todo o corpo docente do curso de Data Science & Big Data da UFPR por sua dedicação e apoio ao longo do curso. Agradeço especialmente ao orientador, Professor Paulo Lisboa, por sua orientação e apoio durante a realização deste estudo.

Agradeço aos meus colegas e amigos, por suas sugestões construtivas e encorajamento ao longo do desenvolvimento deste trabalho.

Por fim, agradeço à minha família por seu amor e paciência durante todo o processo.

Referências

[1] Gama, J., Zliobait˘e, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2013). A Survey on Concept Drift Adaptation. *ACM Computing Surveys*, 46(1), Article 1, 35 pages.

[2] Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in Nonstationary Environments. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5), 1030-1045.

[3] Baena-García, M., Bonilla, J., Bifet, A., Gavaldà, R., & Morales-Bueno, R. (2005). Early Drift Detection Method. In *Proceedings of the Fourth International Workshop on Knowledge Discovery from Data Streams (KDDS)*, 77-86.

[4] Almeida, P. R. L., Oliveira, L. S., Britto Jr., A. S., & Sabourin, R. (2018). Adapting Dynamic Classifier Selection for Concept Drift. *Journal of Machine Learning Research*, 19(1), 1-28.

[5] Albert Bifet and Richard Kirkby. Data stream mining a practical approach. 2009.

[6] Kronberg S., (2024). Concept Drift Detection in Document Classification. *Umeå University, Faculty of Science and Technology, Department of Computing Science*.

[7] Supriya A., Anil K. S., (2022) Concept Drift Detection in Data Stream Mining : A literature review *Journal of King Saud University Computer and Information Sciences*

[8] Souto Maior Barros, R., & Carvalho Santos, S. G. T. (2018). A Large-Scale Comparison of Concept Drift Detectors. *Data Mining and Knowledge Discovery*, 32(3), 778-812.

[9] I. Zliobaite, "How good is the Electricity benchmark for evaluating concept drift adaptation. corr abs/1301,3524 (2013)"2013.

[10] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, “Charac- terizing concept drift,” Data Mining and Knowledge Discovery, vol. 30, no. 4, pp. 964–994, 2016.

[11] Datasets MOA Disponível em: https://moa.cms. waikato.ac.nz/datasets/

[12] Datasets RIVERML Disponível em://riverml.xyz/latest/api/overview

[13] Disponível em: `https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeClassifier.html`