Pontifícia Universidade Católica
do Rio de Janeiro

Ricardo Souza Dias

Detecção de anomalias nas métricas das monitorações de máquinas de um data center

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática da PUC-Rio.

Orientador: Prof. Marcus Vinicius Soledade Poggi de Aragão

Rio de Janeiro
Setembro de 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Ricardo Souza Dias

Detecção de anomalias nas métricas das monitorações de máquinas de um data center

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Marcus Vinicius Soledade Poggi de Aragão
Orientador
Departamento de Informática – PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática – PUC-Rio

Prof. Cassio Freitas Pereira De Almeida
Escola Nacional de Ciências Estatísticas – ENCE

Rio de Janeiro, 13 de Setembro de 2019

Ricardo Souza Dias

Graduou-se em Ciência da Computação pela Universidade Federal da Bahia em junho de 2002. Profissional com 17 anos de experiência em Tecnologia da Informação, com ênfase em Banco de Dados e Desenvolvimento. Atualmente trabalha como Analista Sênior na empresa Globo.com.

Ficha Catalográfica

Souza Dias, Ricardo

Detecção de anomalias nas métricas das monitorações de máquinas de um data center / Ricardo Souza Dias; orientador: Marcus Vinicius Soledade Poggi de Aragão. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 96 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Detecção de Anomalias;. 3. Séries Temporais;. 4. Dados de Streaming;. 5. Algoritmos;. 6. Aprendizagem não supervisionada;. 7. Data Center;. 8. Métricas;. 9. Monitoração.. I. Soledade Poggi de Aragão, Marcus Vinicius. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Ao meu esposo, Adalto Ferreira,
pelo incentivo e apoio.

Agradecimentos

Agradeço a Deus, princípio de tudo, por todas as oportunidades de crescimento que tive nesta vida.

Agradeço aos professores da PUC-Rio, por todos os ensinamentos transmitidos. Em especial ao meu orientador, Professor Marcus Vinicius Soledade Poggi de Aragão, pela oportunidade de ter me acolhido como orientando, pelos ensinamentos, dedicação e sobretudo incentivo.

Agradeço aos Professores Hélio Côrtes Vieira Lopes e Cassio Freitas Pereira de Almeida por terem aceitado participar da banca examinadora.

Agradeço aos amigos, colegas de trabalho e colegas da PUC-Rio, por todo apoio, compreensão e incentivo. Em especial, agradeço ao meu amigo e colega de trabalho, Leopoldo Mauricio, que me ajudou na revisão do texto da dissertação.

Agradeço à minha família, em especial à minha mãe, Maria da Glória Souza Dias, por todo amor, carinho e incentivo e ao meu pai, Raymundo Fernandes Dias, que para mim será sempre uma referência e fonte de inspiração.

Agradeço ao meu esposo, Adalto de Andrade Ferreira, pelo amor e carinho incondicional, pelo apoio e incentivo, fundamental para eu não desistir diante das dificuldades e por me servir de inspiração pela forma como é dedicado aos estudos e trabalho.

Resumo

Souza Dias, Ricardo; Soledade Poggi de Aragão, Marcus Vinicius. **Detecção de anomalias nas métricas das monitorações de máquinas de um data center**. Rio de Janeiro, 2019. 96p. Dissertação de Mestrado– Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Um data center normalmente possui grande quantidade de máquinas com diferentes configurações de hardware. Múltiplas aplicações são executadas e software e hardware são constantemente atualizados. Para evitar a interrupção de aplicações críticas, que podem causar grandes prejuízos financeiros, os administradores de sistemas devem identificar e corrigir as falhas o mais cedo possível. No entanto, a identificação de falhas em data centers de produção muitas vezes ocorre apenas quando as aplicações e serviços já estão indisponíveis. Entre as diferentes causas da detecção tardia de falhas estão o uso técnicas de monitoração baseadas apenas em *thresholds*. O aumento crescente na complexidade de aplicações que são constantemente atualizadas torna difícil a configuração de *thresholds* ótimos para cada métrica e servidor. Este trabalho propõe o uso de técnicas de detecção de anomalias no lugar de técnicas baseadas em *thresholds*. Uma anomalia é um comportamento do sistema que é incomum e significativamente diferente do comportamento normal anterior. Desenvolvemos um algoritmo para detecção de anomalias, chamado DASRS (Decreased Anomaly Score by Repeated Sequence) que analisa em tempo real as métricas coletadas de servidores de um data center de produção. O DASRS apresentou excelentes resultados de acurácia, compatível com os algoritmos do estado da arte, além de tempo de processamento e consumo de memória menores. Por esse motivo, o DASRS atende aos requisitos de processamento em tempo real de um grande volume de dados.

Palavras-chave

Detecção de Anomalias; Séries Temporais; Dados de Streaming; Algoritmos; Aprendizagem não supervisionada; Data Center; Métricas; Monitoração.

Abstract

Souza Dias, Ricardo; Soledade Poggi de Aragão, Marcus Vinicius (Advisor). **Anomaly detection in data center machine monitoring metrics**. Rio de Janeiro, 2019. 96p. Dissertação de mestrado– Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A data center typically has a large number of machines with different hardware configurations. Multiple applications are executed and software and hardware are constantly updated. To avoid disruption of critical applications, which can cause significant financial loss, system administrators should identify and correct failures as early as possible. However, fault-detection in production data centers often occurs only when applications and services are already unavailable. Among the different causes of late fault-detection are the use of thresholds-only monitoring techniques. The increasing complexity of constantly updating applications makes it difficult to set optimal thresholds for each metric and server. This paper proposes the use of anomaly detection techniques in place of thresholds based techniques. An anomaly is a system behavior that is unusual and significantly different from the previous normal behavior. We have developed an anomaly detection algorithm called Decreased Anomaly Score by Repeated Sequence (DASRS) that analyzes real-time metrics collected from servers in a production data center. DASRS has showed excellent accuracy results, compatible with state-of-the-art algorithms, and reduced processing time and memory consumption. For this reason, DASRS meets the real-time processing requirements of a large volume of data.

Keywords

Anomaly Detection; Time Series; Streaming Data; Algorithms; Unsupervised learning; Data Center; Metrics; Monitoring.

Sumário

Lista de figuras

Lista de tabelas

Lista de algoritmos

Lista de abreviaturas

BBB – Big Brother Brasil
CAD – *Contextual Anomaly Detector*
DASRS – *Decreased Anomaly Score by Repeated Sequence*
DBA – *Database Administrator*
DBaaS – *Database as a Service*
ESD – *Extreme Studentized Deviate*
EXPoSE – *EXPected Similarity Estimation*
FN – *False Negative*
FP – *False Positive*
IoT – *Internet of Things*
KNN – *K-Nearest Neighbors*
MAD – *Median Absolute Deviation*
NAB – *Numenta Anomaly Benchmark*
NoSQL – *Not only SQL*
OSE – *Open Source Edition* PaaS – *Platform as a Service*
PCA – *Principal Component Analysis*
RRCT – *Robust Random Cut Forest*
TN – *True Negative*
TP – *True Positive*
VM – *Virtual Machine*

O sucesso é a soma de pequenos esforços,
repetidos dia após dia.

Robert Collier, *Riches Within Your Reach!*.

1
Introdução

1.1
Motivação

Um data center é normalmente composto por milhares de máquinas com diferentes configurações, onde rodam diversas aplicações. O ambiente é bastante dinâmico, onde software e hardware são constantemente atualizados.

A detecção de falhas é uma tarefa essencial em data centers, cujo objetivo é manter os servidores e aplicações de seus clientes o tempo todo disponíveis. Por isso, sistemas de monitoração são implementados para detecção de falhas. Os serviços de monitoração coletam e analisam várias métricas dos servidores; desde as mais gerais como utilização de CPU, memória e tráfego de rede, até as mais específicas como número de sessões de um banco de dados.

A seguir listamos alguns exemplos de problemas que podem causar falhas e que podem ser detectadas através da análise das métricas coletadas dos servidores:

- Aumento inesperado de tráfego de rede provocado por ataques de negação de serviço (*Denial of Service*);
- Aumento ou diminuição no consumo de CPU, no tráfego de rede, no consumo de memória ou acesso ao disco provocado pela atualização de aplicações com código defeituoso (código com *bug*).
- Problemas de hardware, tais como, falhas em interfaces de rede, disco ou memória; configurações inadequadas que causam travamento em processos do sistema operacional, erros de configuração que provocam aumento inesperado no número de sessões do banco de dados, utilização de CPU, tráfego de rede, etc.

No entanto, nem sempre as métricas variam por causa de problemas. Atualizações de software ou hardware muitas vezes provocam alterações esperadas nas métricas coletadas. Além disso, a carga de trabalho pode variar em função da sazonalidade. O consumo de recursos pode ser maior em algumas horas ao longo do dia, em dias de uma semana ou em alguns meses do ano. Aplicações também podem

apresentar maior consumo de recursos à medida que se tornam mais populares. Em todos estes casos ocorrem mudanças no padrão dos valores das métricas coletadas.

Portanto, a detecção de falhas deve ser o mais precisa possível para evitar os prejuízos causados pela indisponibilidade indesejada de serviços. Falsos alarmes também podem causar prejuízos aumentando o custo operacional, quando múltiplos administradores de sistemas precisam ser acionados para analisar problemas inexistentes. Além disso, quando a quantidade de falsos alarmes é elevada, os administradores de sistemas podem deixar de confiar no sistema de monitoração e passar a ignorar os alarmes [Gabel, 2013].

Muitos dos sistemas de detecção de falhas se baseiam em *thresholds*. Assim, um alarme é gerado sempre que o limiar previamente configurado é alcançado. Esse tipo de solução apresenta diferentes tipos de problemas. Definir limiares ótimos é uma tarefa difícil porque eles podem variar em função da carga de trabalho de cada máquina. O limiar precisa ser baixo o suficiente para que falhas não deixem de ser detectadas, ao mesmo tempo que precisa ser alto o suficiente para não gerar falsos alarmes. Além disso, os limiares precisam variar para acompanhar a sazonalidade das cargas de trabalho dos servidores, as atualizações de software ou hardware, etc. Por isso, definir e manter *thresholds* adequados demanda muito tempo dos administradores de sistemas que precisam conhecer profundamente o comportamento dos serviços monitorados. Na prática, os limiares acabam sendo configurados com valores muitos altos. Por isso, muitos alarmes são gerados apenas quando o serviço já se encontra indisponível.

1.2 Objetivos

O objetivo deste trabalho é detectar anomalias, a partir da análise em tempo real, de métricas coletadas de servidores e aplicações. Em outras palavras, as coletas de métricas se traduzem em séries temporais nas quais buscamos determinar trechos que indicam um comportamento incomum e significativamente diferente do comportamento normal anterior.

Em função do grande volume de dados analisados, os algoritmos utilizados devem ser eficientes, apresentando pouco consumo de memória e baixa complexidade e tempo de execução. Caso seja necessário configuração e ajustes de parâmetros, deve ser feito de forma automática devido ao volume e variedade dos servidores e aplicações monitorados. Os algoritmos também precisam se adaptar as possíveis mudanças nos padrões das métricas coletadas e considerar como normal o novo padrão, caso ocorram mudanças.

Definimos um sistema de pontuação para classificar as anomalias encontradas. Cada anomalia identificada recebe uma pontuação que indica a probabilidade

de ser um problema real. A pontuação das anomalias que se repetem é diminuída, assim, elas deixam de ser classificadas como anomalias quando os padrões são frequentes. Dessa forma, as variações de carga de trabalho de servidores, que podem ocorrer em função da sazonalidade, deixam de ser consideradas como anomalias.

Os alarmes gerados a partir das anomalias detectadas devem ser analisados pelos administradores de sistemas que irão investigar a causa da anomalia. Se a mudança do comportamento for gerada por atualizações de software ou hardware, sazonalidade ou algum outro motivo esperado, o administrador deverá ignorar o alarme ao mesmo tempo que o algoritmo deverá reconhecer o novo padrão como normal e não gerar novos alarmes. Caso nenhum dos motivos acima tenha sido identificado, o administrador deverá investigar o que gerou a mudança de comportamento nas métricas do serviço monitorado, que pode ser um dos motivos listados anteriormente como problemas de hardware, *bugs*, ataques de negação de serviço, etc. Neste caso o administrador deverá realizar os procedimentos necessários para sanar o problema.

1.3 Contribuições

Este trabalho propõe e implementa um novo algoritmo de detecção de anomalia chamado DASRS (*Decreased Anomaly Score by Repeated Sequence*). Seu desempenho é comparado com outros algoritmos encontrados no NAB (*Numenta Anomaly Benchmark*) [Ahmad & Lavin, 2015a]. Utilizamos três diferentes *datasets* para comparar a eficiência do algoritmo proposto com a dos encontrados no estado da arte. Utilizamos o *dataset* do NAB, o *dataset* do Yahoo! Webscope [Yahoo! Webscope, 2019] e um *dataset* criado a partir de métricas reais, coletadas de centenas de máquinas do data center da Globo.com[1].

O DASRS obteve ótimos resultados quando comparado com diversos algoritmos do estado da arte encontrados no *benchmark* NAB. Além de obter ótimos resultados de pontuação, o DASRS é mais rápido e apresenta o menor consumo de memória. Por isso, ele possui os requisitos necessários para processar grandes volumes de dados em tempo real.

Implementamos o DASRS no ambiente de produção do data center da Globo.com para analisar em tempo real as métricas coletadas de 2.800 servidores. Esses servidores geram cerca de 14.000 métricas por minuto. Nosso algoritmo analisa e gera uma pontuação de anomalia para as métricas em tempo real.

A criação de um *dataset* com métricas reais de servidores de produção do data center da Globo.com com rótulos de anomalia é outra contribuição deste trabalho.

[1] https://www.globo.com

1.4 Estrutura da Dissertação

O restante desta dissertação está organizado da seguinte forma:

- Capítulo 2: Apresenta os conceitos básicos relacionadas com detecção de anomalias e as definições utilizadas neste trabalho.
- Capítulo 3: Revisa os trabalhos relacionados ao tema de pesquisa.
- Capítulo 4: Descreve o ambiente utilizado no desenvolvimento e avaliação dos algoritmos.
- Capítulo 5: Apresenta os algoritmos de detecção de anomalias proposto neste trabalho.
- Capítulo 6: Mostra os resultados obtidos pelos algoritmos propostos comparando com outros algoritmos do estado da arte.
- Capítulo 7: Apresenta a utilização de um dos algoritmos propostos neste trabalho em uma aplicação real.
- Capítulo 8: Apresenta as conclusões seguidas de uma discussão dos resultados e trabalhos futuros.

2
Conceitos Básicos

Este capítulo apresenta conceitos sobre anomalia, séries temporais, processamento *batch*, processamento *streaming*, mudança de conceito e técnicas de detecção de anomalias. Além disso, apresentamos uma definição para detecção de anomalia.

2.1
Séries Temporais

Os sistemas de monitorações usualmente coletam dados numéricos em frequência uniforme. São números coletados em intervalos regulares de forma sequencial que formam séries temporais.

Definição 2.1 *Uma série temporal X_t é uma sequência de observações de uma variável x_t, ou várias variáveis $(x, y, \dots)_t$, ordenadas no tempo e geralmente ocorrendo em intervalos uniformes. [Brockwell & Davis, 2016] e [Jeffrey, 2008].*

Séries temporais podem ser discretas, quando o intervalo de observações T_0 pertence a um conjunto discreto; ou contínuas, quando as observações são registradas continuamente através de algum intervalo no tempo, por exemplo, quando $T_0 = [0, 1]$.

As séries temporais também podem ser classificadas em univariadas, quando temos apenas uma observação x_t registrada para cada tempo t; ou multivariadas, quando temos duas ou mais observações $(x, y, ...)_t$ registradas para cada tempo t.

A abordagem deste trabalho é dedicada a séries temporais discretas, univariadas e com intervalos uniformes entre as observações.

2.2
Batch versus Streaming

A sequência de números que compõe uma série temporal pode ser disponibilizada no formato *batch* ou *streaming*. *Batch* é um conjunto finito de observações $(x_1, \dots, x_n)$, com $n \in \mathbb{N}$. No processamento *batch*, o algoritmo tem acesso a todo o conjunto de dados e pode utilizá-lo em repetidos passos [Rajaraman & Ullman, 2011]. Os dados em *batch* podem ser disponibilizados de várias formas: arquivos, banco de dados, etc. No processamento *streaming* os dados

chegam em uma sequência, possivelmente infinita, de observações $x_1, x_2, x_3, \ldots$. Neste caso, o algoritmo observa cada $x_t, (1 \leqslant t \leqslant n)$ apenas uma vez e em ordem sequencial.

O processamento *streaming* permite análise continua e em tempo real dos dados. Muitas vezes esse tipo de análise é mandatório para sistemas que monitoram máquinas e aplicações hospedadas em data centers de produção. Por este motivo, os algoritmos estudados e propostos neste trabalho utilizam processamento *streaming*.

Na fase de desenvolvimento e testes realizamos análise de séries temporais utilizando arquivos *batch*. Dessa forma, conseguimos comparar os resultados de diferentes algoritmos (ou variações de um mesmo algoritmo) utilizando um mesmo conjunto de dados (*dataset*). Ainda na etapa de desenvolvimento e testes deste trabalho, apesar de utilizar arquivos *batch*, os dados foram enviados para os algoritmos de forma sequencial, simulando o processamento *streaming*.

2.3 Mudança de Conceito

Mudança de conceito (*concept drift*) refere-se a mudanças nas características dos dados [Žliobaitė et al., 2014]. A mudança de conceito pode ocorrer por diversos motivos, tais como:

- *Upgrades* de *hardware* ou *software*;
- Eventos que provocam aumento ou diminuição significativa de demanda, tais como, início ou fim de jogos de futebol transmitidos pela Internet ou início e fim do programa Big Brother Brasil (BBB)[1]. A análise dos dados obtidos no data center de produção estudado (data center da Globo.com) mostra que ocorre aumento expressivo no acesso às aplicações durante esses eventos;
- Chaveamento do nó *Master* ou Primário de um banco de dados.

Os exemplos descritos são frequentes em data centers de produção e alarmes de anomalias são esperados em função de mudanças de conceito. Mas é fundamental que um algoritmo de detecção de anomalias eficiente seja capaz de identificar e se adaptar a mudanças de conceitos, passando a interpretar como o novo normal os dados com as características novas.

A figura 2.1 ilustra um exemplo de mudança de conceito. O consumo de CPU que inicialmente variava entre 70 e 80, passou a variar entre 10 e 20 depois do instante 500. O consumo de CPU pode ter variado em função de um problema, ou por causa de algum evento programado ou esperado.

[1] https://gshow.globo.com/realities/bbb

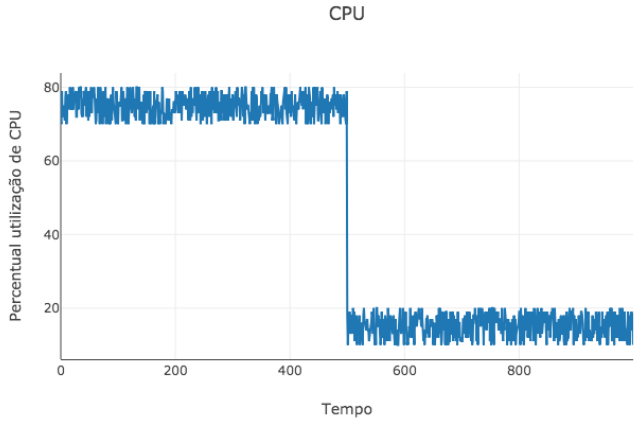


Figura 2.1: Mudança de conceito

2.4 Anomalia

Segundo o Wiktionary[2], anomalia significa "aquilo que apresenta uma anormalidade ou irregularidade, que não é normal". Apesar da existir em ciência da computação uma vasta literatura sobre detecção de anomalias, não existe uma definição única do que é anomalia nem o que é exatamente detecção de anomalia [Schneider et al., 2016]. Neste trabalho iremos adotar a definição elaborada por [Chandola et al., 2009] e estendida por [Ahmad & Lavin, 2015a] para o escopo de séries temporais:

Definição 2.2 *Anomalia é um ponto no tempo em que o comportamento do sistema é incomum e significativamente diferente do comportamento normal anterior.*

Ou seja, anomalias são definidas em relação ao comportamento passado. Assim, um novo comportamento, que em princípio pode ser considerado anormal, deixa de ser anômalo quando persiste, estabelecendo um novo padrão normal.

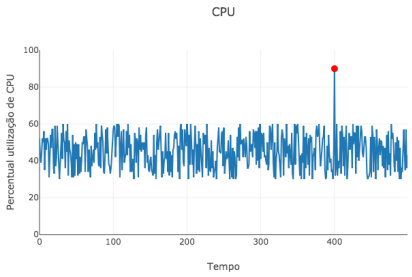
As anomalias podem ser classificadas em três categorias [Chandola et al., 2009]:

– **Anomalias pontuais:** Uma anomalia é pontual quando uma instância de dado pode ser considerada anômala em relação ao restante dos dados, independentemente de onde ela ocorre. Um exemplo de anomalia pontual pode ser observado na figura 2.2a que ilustra o consumo de CPU ao longo do tempo. É possível observar que o consumo de CPU no instante 400 (ponto em vermelho) é significantemente diferente dos demais pontos. Este é o tipo mais simples de anomalia e é o foco da maioria das pesquisas relacionadas a detecção de anomalia.

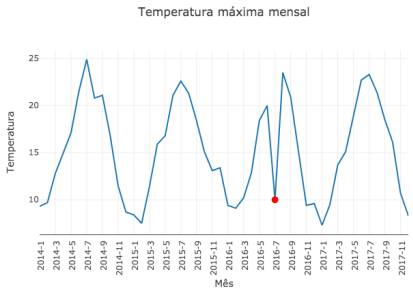
[2]https://pt.wiktionary.org/wiki/anomalia

– **Anomalias contextuais:** Também chamadas de anomalias condicionais, ocorre quando uma instância de dado é anômala apenas em um contexto temporal específico, mas não de outro modo. A figura 2.2b mostra a temperatura máxima mensal entre os anos de 2014 e 2017. Embora a temperatura indicada no ponto vermelho possa ser considerada normal para meses entre dezembro e março, ela é anormal para o mês de julho. Trata-se, portanto, de uma anomalia contextual. A noção do contexto é induzida pela estrutura do dado. Cada instância de dado é definida usando dois conjuntos de atributos: atributos contextuais e atributos comportamentais. Os atributos contextuais determinam o contexto da instância. Nas séries temporais, o tempo é o atributo contextual, pois determina a posição da instância em relação a sequência completa. Os atributos comportamentais definem as características da instância que não dependem do contexto. Nas séries temporais univariadas, o atributo comportamental é o dado numérico da série e seu significado depende do problema analisado, podendo representar, por exemplo: temperatura, consumo de CPU, número de requisições, sessões de um banco de dados, etc.

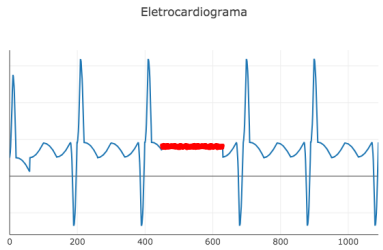
– **Anomalias coletivas:** Se um grupo de instâncias de dados relacionados for anômalo em relação ao conjunto de dados inteiro, este grupo será denominado de anomalia coletiva. Embora cada instância específica desse grupo não precise ser anômala, sua ocorrência coletiva é anômala. A figura 2.2c ilustra um eletrocardiograma humano. A região em vermelho corresponde a uma anomalia coletiva. Apesar de ocorrer por um longo período de tempo a anomalia coletiva observada durante o eletrocardiograma não representa um comportamento considerado normal, ilustrado no restante do gráfico.



(a) Anomalia Pontual



(b) Anomalia Contextual



(c) Anomalia Coletiva

Figura 2.2: Tipos de anomalias.

Anomalias pontuais podem ocorrer em qualquer conjunto de dados. Por outro lado, as anomalias coletivas ocorrerem apenas em conjuntos de dados cujas instâncias estão relacionadas. A respeito das anomalias contextuais, sua ocorrência depende da disponibilidade dos atributos de contexto. Portanto, tanto uma anomalia pontual quanto uma anomalia coletiva podem ser do tipo contextual se analisadas em relação a um contexto específico [Chandola et al., 2009].

2.5 Técnicas de Detecção de Anomalia

As técnicas de detecção de anomalia muitas vezes dependem da existência ou não de dados devidamente catalogados, informando se estes são considerados anômalos ou não. Geralmente as anomalias são catalogadas de forma manual por um especialista humano e requer um esforço substancial. Por isso, a obtenção de dados rotulados, precisos e representativos de todos os tipos de comportamento, é muitas vezes proibitivamente caro. Alternativamente, um conjunto de treinamento apenas com dados normais é muito mais fácil de obter do que o conjunto completo que cubra todos os comportamentos anômalos. Dependendo da disponibilidade e do tipo de anotações (rótulos) disponíveis, os algoritmos de detecção de anomalias se dividem em três categorias [Chandola et al., 2009]:

– **Supervisionado:** As técnicas de detecção de anomalia supervisionada exi-

gem um conjunto de dados rotulados como normais ou anômalos. A abordagem típica em tais casos envolve o treinamento e construção de um classificador para as classes normais e anômalas. As instâncias de dados são comparadas com o modelo para determinar a qual classe cada uma pertence.

– **Semi-supervisionado:** As técnicas de detecção de anomalia semi-supervisionadas assumem que o conjunto de treinamento possui apenas os tipos de dados da classe "normal". Portanto, a abordagem típica usada nesta técnica envolve a construção de um modelo preditivo para a classe normal e utilização deste modelo para identificar as instâncias de dados que são anômalas.

– **Não Supervisionado:** As técnicas de detecção de anomalia não supervisionada não requerem dados de treinamento. Os algoritmos de detecção de anomalias desta categoria partem do princípio que as ocorrências de anomalias são eventos raros. Por isso, eles procuram instâncias de dados que não se encaixem com o restante do conjunto de dados.

Devido a impossibilidade de obter um conjunto preciso e representativo de todos os tipos de comportamento, iremos utilizar neste trabalho técnicas não supervisionadas de detecção de anomalias. Os dados rotulados utilizados no ambiente de testes deste trabalho não são utilizados para treinamento e sim para verificar a acurácia dos algoritmos analisados.

2.6 Detecção de Anomalias

O principal objetivo deste trabalho, detecção de anomalias em séries temporais, é definido abaixo:

Definição 2.3 *A detecção de anomalias em séries temporais consiste em analisar a sequência de números que formam a série temporal e para cada observação x_t gerar uma predição $P(x_t)$ de anomalia.*

A predição de anomalia pode ser uma pontuação de anomalia ou um rótulo (normal ou anômalo) [Chandola et al., 2009]. A utilização de pontuação dá mais flexibilidade ao analista que deseja analisar apenas as anomalias com maior pontuação ou criar um limiar (*threshold*) para selecionar anomalias que ele considera mais "importantes". Além disso, os algoritmos que geram predição de pontuação podem ser facilmente convertidos para gerar rótulos. A maioria dos algoritmos estudados e desenvolvidos neste trabalho utilizam pontuação.

3
Trabalhos Relacionados

Neste trabalho, decidimos aplicar técnicas de detecção de anomalia sobre as métricas que são coletadas a partir da monitoração dos servidores de um data center de produção. Entre as técnicas pesquisadas estão as que realizam processamento *streaming* de séries temporais, aprendizado de máquina não supervisionado e que ajustes de parâmetros, quando necessário, sejam feitos de forma automática.

A detecção de anomalias em séries temporais é uma área bastante estudada. O primeiro trabalho sobre este tema foi apresentado por Fox [Fox, 1972]. Chandola et al. [Chandola et al., 2009] compilaram uma abrangente pesquisa sobre detecção de anomalia, não restrita a séries temporais. Os autores propõem o agrupamento das técnicas de detecção de anomalias em diferentes categorias, em função da abordagem central de cada técnica.

Em [Ahmad & Lavin, 2015a] os autores propõem um *framework* para comparar algoritmos de detecção de anomalias em séries temporais. O *framework* desenvolvido, que recebeu o nome de NAB (Numenta Anomaly Benchmark), fornece um conjunto de ferramentas para testar, medir e comparar diferentes algoritmos de detecção de anomalias. Por ser um *framework* utilizado em diversas pesquisas, decidimos utilizar o NAB para avaliar os algoritmos desenvolvidos nesta dissertação. Assim, comparamos os resultados dos algoritmos propostos com os encontrados no *benchmark* do NAB.

Em [Ahmad & Purdy, 2016] e [Ahmad et al., 2017] é apresentado o algoritmo Numenta HTM (*Hierarchical Temporal Memory*), cuja técnica de detecção de anomalia é baseada na memória temporal hierárquica. As sequencias temporais de dados são modeladas pelo HTM e para cada momento t o algoritmo faz várias previsões para o valor do dado para o momento $t + 1$. Tais previsões são então comparadas com o valor real obtido a fim de determinar se ele pode ser considerado normal ou anômalo.

Smirnov [Smirnov, 2016] desenvolveu o algoritmo CAD (Contextual Anomaly Detector). Nele, as observações de entrada são modeladas e armazenadas em estruturadas de dados internas. A cada observação de entrada o CAD consulta suas estruturas internas e gera uma pontuação (*escore*) de anomalia. Assim, ele busca indicar o quanto um valor de entrada observado corresponde ou não a um novo

contexto em função das observações precedentes.

Burnaev e Ishimtsev [Burnaev & Ishimtsev, 2016] propõem algoritmos de detecção de anomalias baseados em distância e densidade. Os algoritmos executam métodos de extração de características (*features*). Em seguida, definem uma pontuação quando o novo valor difere significativamente dos dados previamente observados. A partir daí, os algoritmos realizam uma interpretação probabilística da pontuação.

Em [Guha et al. 2016] uma técnica de detecção de anomalias baseada em cortes aleatórios de estruturas de dados em árvores é apresentada. Guha et al. propõem um novo método, chamado RRCT (Robust Random Cut Forest), que identifica uma estrutura robusta de dados de corte aleatório que pode ser usada como esboço do fluxo de entrada. A anomalia é baseada na influência de uma observação ainda não vista sobre o restante dos dados.

Wang et al. [Wang et al. 2011], por outro lado, propõem técnicas de detecção de anomalias baseadas nas estatísticas Tukey e Entropia Relativa, enquanto Schneider et al. [Schneider et al., 2016] implementa o algoritmo EXPoSE (EXPected Similarity Estimation), baseado em kernel e capaz de calcular eficientemente a similaridade entre novos pontos de dados e a distribuição de dados regulares.

Kejariwal [Kejariwal, 2015] desenvolve um algoritmo que usa uma combinação de técnicas estatísticas para detecção de anomalias. O teste ESD (Extreme Studentized Deviate) generalizado [Rosner, 1983] é combinado com métricas estatísticas robustas e a aproximação por partes é usada para detectar tendências de longo prazo. O algoritmo foi projetado especificamente para detectar anomalias sazonais no contexto de dados de redes sociais e funciona bem quando surgem anomalias em dados periódicos que não são muito diferentes dos dados anteriores. Contudo, os resultados não são bons quando uma tendência de série temporal muda ao longo do tempo.

Skyline [Etsy, Inc., 2013] é um sistema de detecção de anomalias em tempo real. O algoritmo implementado utiliza um conjunto de técnicas estatísticas de detecção de anomalias e um esquema de votação para gerar a pontuação final da anomalia. As técnicas estatísticas incluem desvio absoluto mediano, média da primeira hora, desvio padrão da média, desvio padrão da média móvel, subtração da média acumulada, mínimos quadrados e caixas de histograma. O algoritmo utiliza a técnica de janela deslizante. Dessa forma, os testes estatísticos são feitos apenas com as últimas observações analisadas.

O projeto Skyline foi descontinuado pela empresa que o desenvolveu. No entanto, o projeto Earthgecko Skyline [Earthgecko, 2019], que foi criado a partir do Skyline, implementa melhorias sobre o projeto original. O Earthgecko Skyline mantém o núcleo Skyline que implementa um conjunto de técnicas estatísticas de

detecção de anomalias e um esquema de votação para gerar a pontuação final da anomalia. Entretanto, técnicas para aumentar o desempenho dos algoritmos são adicionadas, além disso, a lógica do esquema de votação é otimizada.

Adams e MacKay [Adams & MacKay 2007] propõem um modelo de ponto de mudança alternativo que pode fazer inferências eficientes. Os pesquisadores reformularam o problema do ponto de mudança para ser mais apropriado para cálculo: em vez de se preocupar em colocar retrospectivamente pontos de mudança, o algoritmo é interessado na mudança mais recente.

4
Ambiente de Desenvolvimento e Testes

Para avaliação e testes dos algoritmos desenvolvidos neste trabalho, utilizamos o *framework* NAB (*Numenta Anomaly Benchmark*) [Ahmad & Lavin, 2015a] lançado por Numenta[1].

Este capítulo descreve o *framework* NAB, seu *dataset*, o cálculo da pontuação e os algoritmos incluídos no *framework*. Além do *dataset* incluído no NAB, foram utilizados outros dois *datasets* cuja descrição se encontra no final deste capítulo.

4.1
Framework NAB

O *framework* NAB fornece um conjunto de ferramentas para testar, medir e comparar diferentes algoritmos de detecção de anomalias, com dados *streaming*, num ambiente controlado e repetível. Contém código, algoritmos de detecção de anomalia, documentação, séries temporais de diferentes campos e *benchmark* comparativo dos algoritmos. NAB consiste em um repositório de código aberto, escrito em Python[2] e disponível no GitHub[3].

O *framework* permite que o desenvolvedor escreva seu algoritmo de detecção de anomalias e possa medir seu desempenho através da pontuação calculada pelo NAB. Além disso, permite comparar o resultado do seu algoritmo com outros algoritmos através do *benchmark* incluído no *framework*.

A figura 4.1 ilustra o funcionamento do NAB. O *dataset* X é um conjunto de arquivos, cada um corresponde a uma série temporal e contém informações das observações (data e hora e valor escalar). As observações são enviadas sequencialmente para o detector Y, simulando o processamento *streaming*. Para cada observação, o detector deve gerar um escore de anomalia entre 0 e 1. O escore de anomalia, através do *threshold* de anomalia, é convertido num valor discreto indicando a presença ou ausência de anomalia. Opcionalmente, através do módulo de otimização, o NAB pode calcular o *threshold* que irá gerar a maior pontuação para um determinado perfil de aplicação. O cálculo da pontuação NAB bruta é feito utilizando as seguintes informações: *dataset* X, rótulos reais de anomalias do *dataset* X, es-

[1] https://numenta.org
[2] https://www.python.org/
[3] https://github.com/numenta/NAB

cores de anomalia gerados pelo detector Y, *threshold* de anomalia do detector Y e perfil de aplicação Z. Por fim, a pontuação NAB bruta é normalizada e no cálculo desta pontuação utiliza-se a pontuação gerada pelos detectores "perfeito" e "nulo". A pontuação NAB normalizada é a que utilizada no *benchmark* de pontuação. Nas próximas seções iremos explicar com mais detalhes estes componentes.

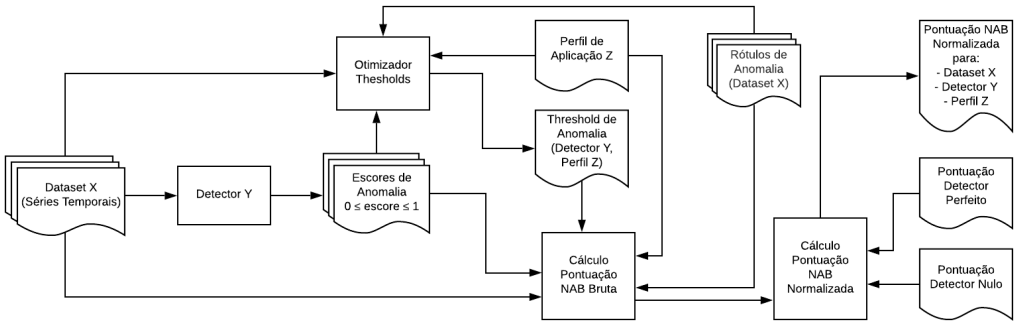


Figura 4.1: Framework NAB

4.2 Descrição do Dataset do NAB

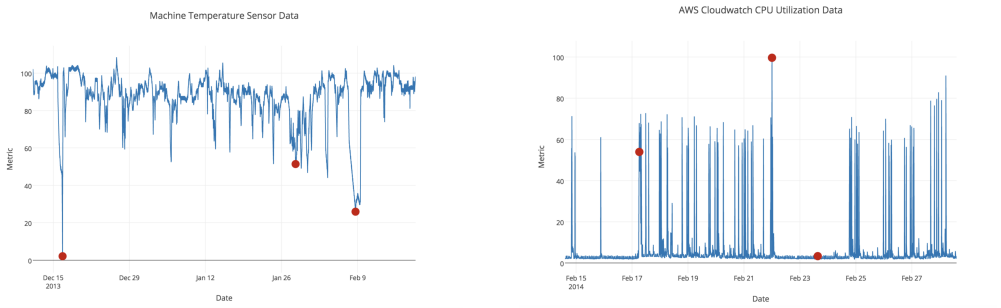
O *dataset* NAB foi construído com o objetivo de apresentar aos algoritmos os desafios que enfrentarão em cenários do mundo real: uma combinação de anomalias pontuais e contextuais, dados com e sem ruído e fluxo de dados cujas características evoluam com o tempo.

Consiste de 58 séries temporais univariadas, cada série é armazenada em um arquivo e cada linha contém registro de data e hora, além de um valor escalar. Cada série possui entre 1.000 e 22.000 observações, totalizando 365.551 observações. Abrange métricas de vários domínios, incluindo tráfego rodoviário, utilização da CPU, rede e métricas de mídias sociais. Alguns arquivos foram gerados artificialmente, mas a maioria foi criado a partir de dados coletados do mundo real. Cinco arquivos do *dataset* não contêm anomalias e os demais arquivos contêm uma ou mais anomalias. A descrição detalhada dos arquivos se encontra no *readme* do diretório onde está localizado o *dataset*[4].

Além do *dataset* das séries de dados, o NAB também fornece rótulos de referência com as anomalias presentes. Os rótulos foram gerados manualmente pela equipe que criou o *framework* NAB, observando individualmente as séries temporais e inclui situações reais em que a causa da anomalia foi identificada. O procedimento para criação destes rótulos está documentado em [Numenta, Inc., 2019.a].

As figuras 4.2a e 4.2b são dois exemplos representativos de séries temporais incluídas no NAB. Os pontos em vermelho são as anomalias de referência.

[4]https://github.com/numenta/NAB/tree/master/data



(a) Exemplo 1 (b) Exemplo 2

Figura 4.2: Dois exemplos de séries temporais do NAB. Figura extraída de [Ahmad & Lavin, 2015a]

4.3 Cálculo da Pontuação NAB

A medida de desempenho proposta pelos autores do NAB aos algoritmos testados leva em consideração a resposta do detector em identificar de forma correta as anomalias e também a antecedência em que a anomalia é detectada. Ou seja, o detector é recompensado pelas anomalias detectadas corretamente (quanto mais cedo a anomalia é detectada, maior a recompensa) e penalizado pelas anomalias não detectadas ou anomalias detectadas de forma incorreta (anomalias não existentes). Os autores definiram um conjunto de regras para o cálculo do escore, cujos aspectos principais são: janelas de anomalia, função de pontuação e os utilização de perfis de aplicação. Estes aspectos são descritos abaixo e a discussão detalhada pode ser encontrada em [Ahmad & Lavin, 2015a].

Para poder incorporar pontos na detecção precoce da anomalia, foi introduzido o conceito de janelas de anomalia. As janelas representam um intervalo de pontos de dados que é centralizado em torno de um rótulo de anomalia de verdade. A figura 4.3 mostra um exemplo usando os dados da figura 4.2a. A região sombreada na cor vermelha representa as janelas de anomalia. O tamanho de cada janela de anomalia corresponde a 10% do tamanho do arquivo dividido pelo número anomalias presentes no arquivo. A região sombreada na cor roxa representa o período probatório e corresponde a 15% dos dados iniciais do arquivo. Durante este período, o detector pode aprender os padrões de dados sem ser avaliado.

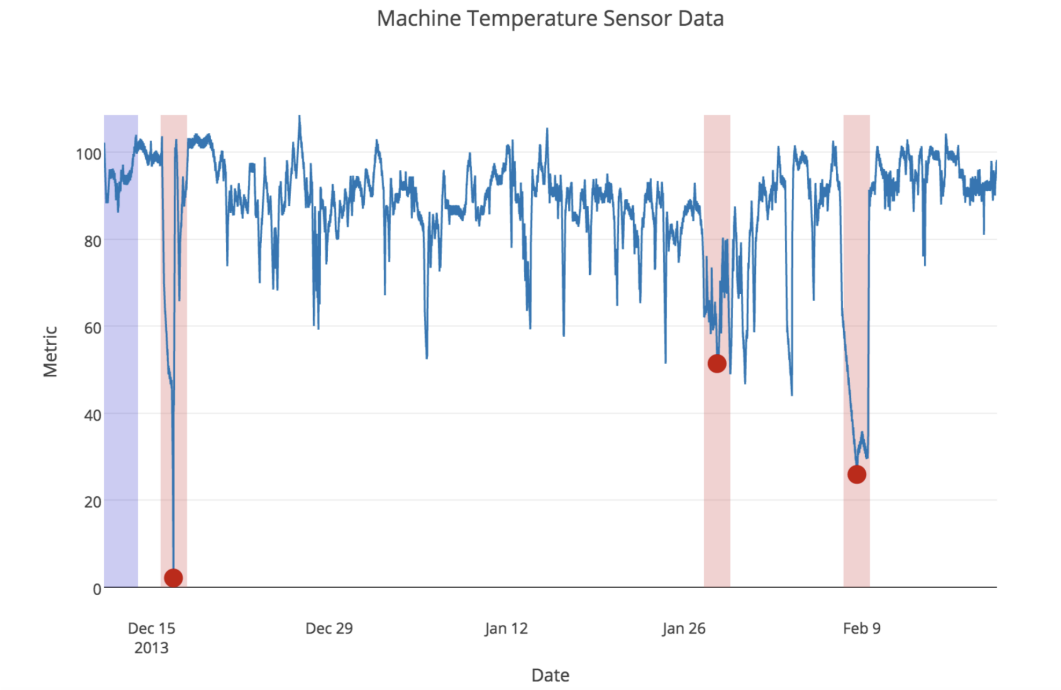


Figura 4.3: Exemplo de série temporal do NAB com janelas de anomalia e período probatório. Figura extraída de [Ahmad & Lavin, 2015a]

Quando um algoritmo é testado no NAB, as detecções resultantes são utilizadas no cálculo da pontuação do algoritmo. A função de pontuação é aplicada em cada janela para identificar e avaliar as detecções feitas de forma correta (*True Positive* - TP), as detecções incorretas (*False Positive* - FP) e as anomalias não detectadas (*False Negative* - FN). As detecções em uma janela identificam corretamente dados anômalos e aumentam a pontuação no NAB. Se ocorrer uma detecção no início de uma janela, será dado maior valor pela função de pontuação do que uma detecção no final da janela. Detecções múltiplas dentro de uma única janela identificam os mesmos dados anômalos, portanto, é usado somente a detecção mais antiga (mais valiosa) para a contribuição da pontuação. As detecções que estão fora das janelas são falsos positivos (FP), dando uma contribuição negativa para a pontuação do NAB. O local onde o FP é identificado também é usado na função de pontuação, de modo que um FP que ocorre logo após uma janela prejudica menos a pontuação do NAB que um FP que ocorre longe da janela. Perder uma janela completamente, ou um falso negativo (FN), resulta em uma contribuição negativa para a pontuação. Os pontos fora das janelas que não tiveram anomalias identificados, ou seja, verdadeiro negativo (*True Negative* - TN) não contribuem para a pontuação.

A figura 4.4 mostra um exemplo de como a função sigmóide (mostrada no final desta seção) é usada no cálculo do escore. O primeiro ponto é um FP que precede a janela de anomalia (linhas tracejadas vermelhas) e contribui com -1.0 para a pontuação. Dentro da janela, vemos duas detecções e contamos apenas o TP

mais antigo para a pontuação. Existem dois FPs após a janela. O primeiro é menos prejudicial porque está próximo da janela e o segundo rende -1.0 porque está muito longe da janela para ser associada com uma anomalia verdadeira.

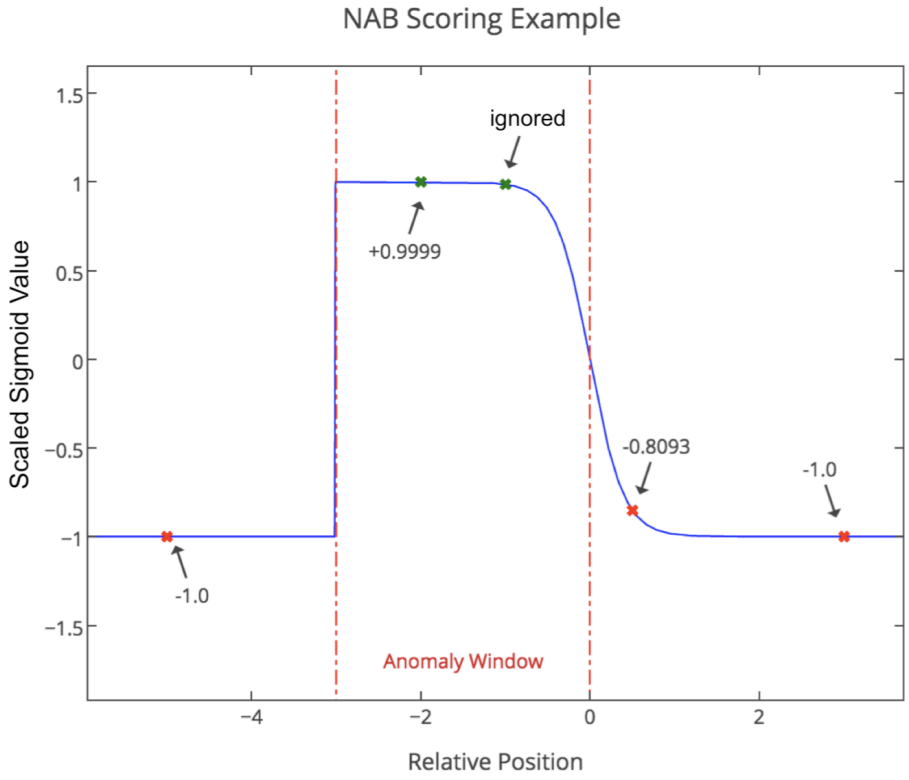


Figura 4.4: Cálculo do escore do NAB. Figura extraída de [Ahmad et al., 2017]

Os perfis de aplicação definem pesos aos falsos positivos e falsos negativos para atender cenários em que menos detecções perdidas ou menos detecções erradas são mais valorizadas. O NAB inclui três diferentes perfis de aplicação: padrão, recompensa poucos FPs e recompensa poucos FNs. O perfil padrão atribui TPs, FPs, TNs e FNs com pesos relativos, de forma que as detecções aleatórias feitas em 10% do tempo obteriam uma pontuação final zero em média. Os dois últimos perfis definem penalidades maiores para FPs e FNs, respectivamente. Estes dois perfis são um pouco arbitrários, mas foram incluídos no projeto para que os usuários possam ajustar os pesos de acordo com as necessidades específicas da aplicação. A tabela 4.1 mostra os pesos propostos pelo NAB para TP, FP, TN e FN para cada um dos perfis.

Tabela 4.1: Perfis de aplicação propostos pelo NAB

	Pesos			
Perfil	TP	FP	TN	FN
Padrão	1.0	-0.11	1.0	-1.0
Recompensa Poucos FPs	1.0	-0.21	1.0	-1.0
Recompensa Poucos FNs	1.0	-0.11	1.0	-2.0

A combinação de janelas de anomalia, função de pontuação e perfis de aplicação permitem que os pesquisadores avaliem o desempenho das implementações do detector de anomalias em relação aos requisitos do detector ideal para sua aplicação.

A pontuação final do NAB para um determinado algoritmo e um determinado perfil é calculada da seguinte maneira. Seja A o perfil em consideração e $A_{TP}, A_{FP}, A_{FN}, A_{TN}$ os pesos correspondentes a TP, FP, FN, TN. Estes pesos estão limitados a $0 \leq A_{TP}, A_{TN} \leq 1$ e $-1 \leq A_{FP}, A_{FN} \leq 0$. Seja D o conjunto de arquivos de dados e Y_d o conjunto de instâncias de anomalias detectadas para o arquivo de dados d. Conforme citado anteriormente, se o algoritmo identifica múltiplas anomalias na mesma janela, somente a primeira é levada em consideração. O número de janelas com zero detecções neste arquivo é o número de falsos negativos, representados por f_d.

A seguinte função sigmoidal define o peso das detecções individuais, dada uma janela de anomalia e a posição relativa de cada detecção:

$$\sigma^A(y) = (A_{TP} - A_{FP})\left(\frac{1}{1+e^{5y}}\right) - 1 \tag{4-1}$$

onde y é a posição relativa da detecção dentro da janela de anomalia. As detecções dentro da janela (TP) contribuem com uma pontuação positiva no cálculo do escore enquanto que as detecções fora da janela (FP) contribuem com uma pontuação negativa. A função é projetada de tal forma que os TPs detectados no início da janela contribuem com mais pontos que as detecções feitas no final da janela. Da mesma forma, os FPs próximos a janela contribuem com pontuações negativas menores do que as detecções mais distantes da janela.

Faltar completamente uma janela é contado como um falso negativo e atribuiu uma pontuação de A_{FN}. A pontuação de um arquivo de dados é a soma das pontuações das detecções individuais mais o peso das janelas que não tiveram anomalias detectadas:

$$S_d^A = \left(\sum_{y \in Y_d} \sigma^A(y)\right) + A_{FN} f_d \tag{4-2}$$

A equação 4-2 acumula a pontuação ponderada de cada verdadeiro positivo e falso positivo, abatendo com uma contagem ponderada de todos os falsos negativos. A pontuação bruta de referência para um determinado algoritmo é simplesmente a soma das pontuações brutas de todos os arquivos de dados:

$$S^A = \sum_{d \in D} S_d^A \tag{4-3}$$

A pontuação final NAB relatada é uma pontuação normalizada calculada da seguinte forma:

$$S_{NAB}^A = 100 \times \frac{S^A - S_{null}^A}{S_{perfect}^A - S_{null}^A} \tag{4-4}$$

onde $S_{perfect}^A$ é a pontuação obtida por um detector "perfeito" (um que produz todos os verdadeiros positivos no início da janela e nenhum falso positivo) e S_{null}^A é a pontuação obtida por um detector "nulo" (um que não produz nenhuma detecção de anomalia). Segue da Equação 4-4 que a pontuação máxima (normalizada) que um detector pode alcançar no NAB é 100 e um algoritmo que não faz nenhuma detecção terá pontuação 0.

A implementação das funções descritas nesta seção está disponível no projeto NAB[5].

4.4 Benchmark de Desempenho

O repositório do NAB contém diversos algoritmos de detecção de anomalias, todos de código aberto e alguns deles utilizados em aplicações comerciais. Os autores do NAB criaram um *benchmark* para medir o desempenho destes algoritmos e acrescentaram dois detectores ao *benchmark* que não estão presentes no repositório de algoritmos. O primeiro, Random Cut Forest, é um código proprietário da AWS[6]. O segundo, Twitter ADVec, apesar de ser *open source*, sua implementação é em R[7] e não em Python[8]. Os primeiros colocados deste *benchmark* representam o estado da arte para esta categoria de algoritmos.

A tabela 4.2 contém o *benchmark* de desempenho dos algoritmos de detecção de anomalia publicado na página do repositório do NAB[9]. A descrição das características principais destes algoritmos é mostrada em seguida.

[5]https://github.com/numenta/NAB
[6]https://aws.amazon.com
[7]https://www.r-project.org
[8]https://www.python.org
[9]https://github.com/numenta/NAB

Tabela 4.2: Benchmark dos Detectores Incluídos no Repositório do NAB

	Pontuação por Perfil de Aplicação		
Detector	**Padrão**	**Recompensa Poucos FPs**	**Recompensa Poucos FNs**
Numenta HTM	70.5-69.7	62.6-61.7	75.2-74.2
CAD OSE	69.9	67.0	73.2
earthgecko Skyline	58.2	46.2	63.9
KNN CAD	58.0	43.4	64.8
Relative Entropy	54.6	47.6	58.8
Random Cut Forest	51.7	38.4	59.7
Twitter ADVec	47.1	33.6	53.5
Windowed Gaussian	39.6	20.9	47.4
Etsy Skyline	35.7	27.1	44.5
Bayesian Changepoint	17.7	3.2	32.2
EXPoSE	16.4	3.2	26.9

4.5 Detectores mais Relevantes do Benchmark do NAB

Nesta seção descrevemos os detectores que obtiveram maior pontuação no *benchmark* do NAB. São eles: Numenta HTM, CAD OSE e earthgecko Skyline.

4.5.1 Numenta HTM

O detector Numenta HTM [Ahmad & Purdy, 2016] [Ahmad et al., 2017] foi criado pela empresa Numenta[10], a mesma que desenvolveu o NAB. O algoritmo Numenta HTM utiliza Memória Temporal Hierárquica (HTM, do inglês *Hierarchical Temporal Memory*) para detecção de anomalia. O HTM é uma tecnologia de aprendizado de máquina inspirada da estrutura do neocórtex [Hawkins & Blakeslee, 2004], [Hawkins & Ahmad, 2016], [Ahmad & Hawkins, 2016].

Dado um fluxo de dados em tempo real $\ldots, x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}, \ldots$, o algoritmo modela as sequências temporais deste fluxo. A figura 4.5 mostra as etapas executadas pelo algoritmo. Na primeira etapa, o HTM mapeia a entrada x_t, gerando $a(x_t)$ e $\pi(x_t)$. $a(x_t)$ é uma codificação esparsa da entrada atual e $\pi(x_t)$ é um vetor esparso que representa as predições feitas pelo HTM para $a(x_{t+1})$, ou seja, as predições para a próxima observação x_{t+1}. A dimensão de ambos os vetores é igual ao número de colunas na rede HTM. Nas etapas seguintes, o algoritmo calcula o valor S_t (*prediction error*), também chamado de escore bruto de anomalia, e o L_t (*anomaly likelihood*), que é o escore final de anomalia gerado pelo algoritmo. O cálculo do S_t e L_t serão detalhados abaixo.

[10]https://numenta.com

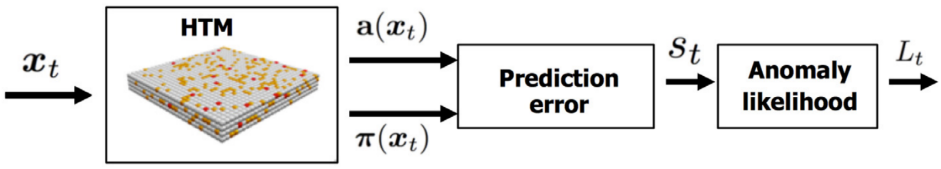


Figura 4.5: Etapas do algoritmo Numenta HTM. Figura extraída de [Ahmad et al., 2017]

4.5.1.1 Cálculo do Prediction Error

O *prediction error*, S_t, é um valor escalar entre 0 e 1, dado pela fórmula abaixo:

$$S_t = 1 - \frac{\pi(x_{t-1}) \times a(x_t)}{|a(x_t)|} \tag{4-5}$$

onde $|a(x_t)|$ é o módulo do vetor $a(x_t)$, ou seja, seu comprimento. O erro S_t será 0 se $a(x_t)$ combinar perfeitamente com a previsão e 1 se os dois vetores forem ortogonais.

Esta pontuação é uma medida de quão bem o modelo HTM previu o padrão para a entrada atual x_t. Dependendo dos dados e, particularmente, da quantidade de ruído presente, essa pontuação de anomalia pode produzir muitos falsos positivos. Então, ao invés de utilizar o escore bruto de anomalia diretamente, o escore final de anomalia (L_t) é calculado a partir da distribuição de S_t.

4.5.1.2 Cálculo do Anomaly Likelihood

Anomaly likelihood, L_t, é uma métrica probabilística definindo o quanto anômalo o estado atual é, baseado no histórico de predição do modelo HTM. O algoritmo mantém uma janela com os últimos W valores de erro. A média (μ_t) e a variância (σ_t^2) da amostra são continuamente atualizadas a partir dos erros recentes, como mostrado nas equações abaixo:

$$\mu_t = \frac{\sum_{i=0}^{W-1} S_{t-i}}{W} \tag{4-6}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{W-1} (S_{t-i} - \mu_t)^2}{W - 1} \tag{4-7}$$

Para calcular L_t, são utilizados a função Q [Karagiannidis & Lioumpas, 2007] e a média recente de curto prazo de erros de predição. L_t é o complemento da função Q, dado pela fórmula abaixo:

$$L_t = 1 - Q\left(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}\right) \tag{4-8}$$

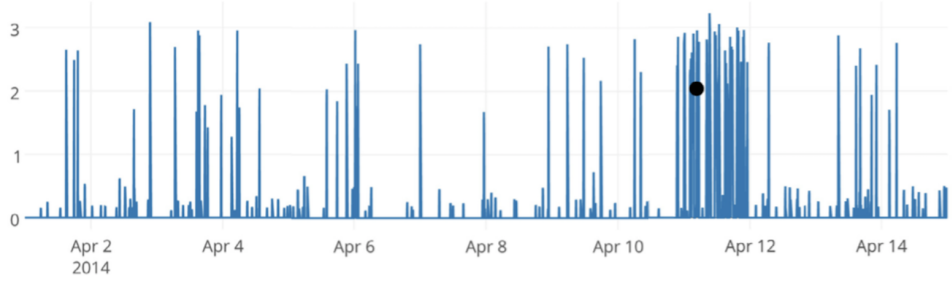
onde,

$$\tilde{\mu_t} = \frac{\sum_{i=0}^{W'-1} S_{t-i}}{W'} \tag{4-9}$$

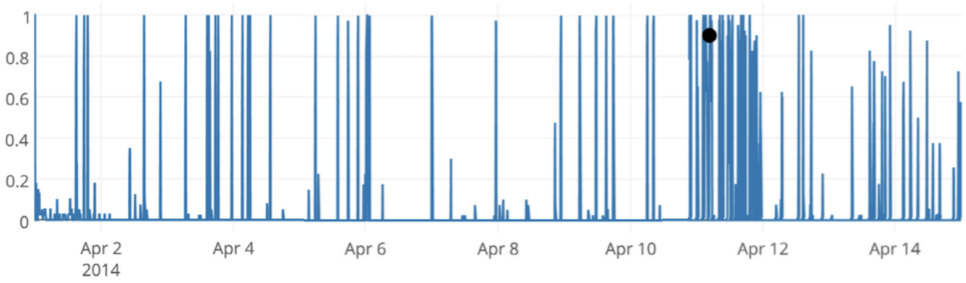
W' é a janela para a média deslizante de curto prazo, onde $W' << W$, a duração para calcular a distribuição de erros de predição.

É importante notar que L_t é baseado na distribuição de S_t, não na distribuição de x_t. Em cenários previsíveis e sem ruídos, L_t comporta-se de forma semelhante a S_t. Nesses casos, a distribuição de erros terá uma variação muito pequena e será centralizada perto de 0. Qualquer pico em S_t levará de forma similar a um pico correspondente em L_t. No entanto, em cenários com alguma aleatoriedade ou ruído inerente, a variação será mais ampla e a média maior que 0.

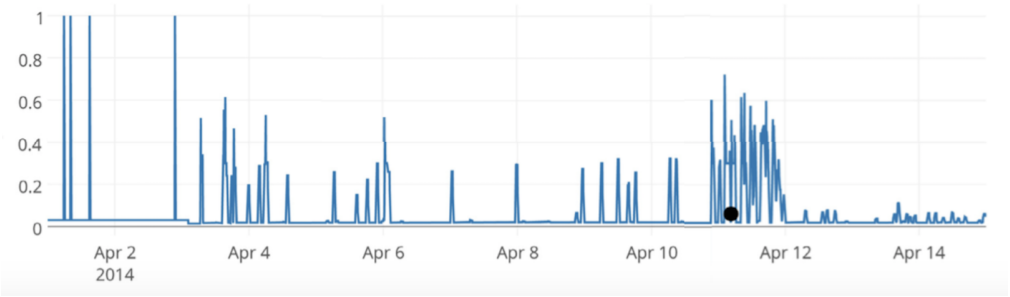
A figura abaixo mostra uma das séries de dados do *dataset* NAB juntamente com o *prediction error* e *anomaly likelihood* calculados pelo algoritmo Numenta HTM. A série da figura 4.6a mostra métrica de latência e o ponto preto mostrado nos 3 gráficos corresponde a uma anomalia presente nos rótulos de anomalia desta série.



(a) Latência



(b) Prediction error



(c) Anomaly likelihood

Figura 4.6: Exemplo de série temporal com correspondente *prediction error* e *anomaly likelihood*. Figura extraída de [Ahmad et al., 2017]

4.5.1.3 Anomalias Pontuais

Apesar de não está descrito em nenhum dos dois artigos publicados sobre detector Numenta HTM [Ahmad & Purdy, 2016] [Ahmad et al., 2017], no código fonte do algoritmo[11] tem um trecho pra detectar anomalias pontuais (no algoritmo, é chamado anomalia espacial) e uma vez detectada este tipo de anomalia, o escore de anomalia retornado é 1, independente do escore de anomalia S_t ou Lt.

[11]https://github.com/numenta/NAB/blob/master/nab/detectors/numenta/numenta_detector.py

4.5.2 CAD OSE

O detector CAD OSE (Contextual Anomaly Detector - Open Source Edition) [Smirnov, 2016] foi o vencedor da competição NAB de 2016 para detecção de anomalia em tempo real [Numenta, Inc., 2016]. Até o momento, o autor do algoritmo não publicou nenhum artigo sobre o mesmo, mas o código fonte está disponível no repositório do NAB[12] e no repositório GIT do autor[13]. Por esse motivo, analisamos o código do detector CAD OSE para descrever seu funcionamento.

As observações de entrada são modeladas e armazenadas em estruturadas de dados internas. A modelagem consiste em uma série de transformações que incluem a normalização do dado, representação binária do dado normalizado e mapeamento em estruturas de dados internas chamada $contexto$. Em cada momento t, o algoritmo compara os elementos do contexto corrente (instante t) com os elementos dos contextos anteriores (até o instante $t-1$), atualiza variáveis de dados internas e calcula o escore bruto de anomalia dado pela equação abaixo:

$$rawAnomalyScore = \frac{1 - \frac{numActiveContexts}{numSelContexts} + \frac{numNewCont}{numUniqPotNewContext}}{2} \quad (4\text{-}10)$$

onde $numActiveContexts$ é a quantidade de elementos que coincidem entre o contexto corrente e o último contexto analisado ($t-1$), $numSelContexts$ é a quantidade de elementos da interseção entre os elementos do último contexto ($t-1$) e os elementos dos contextos anteriores, $numNewCont$ é número de elementos novos adicionados ao contexto e $numUniqPotNewContext$ é a quantidade de elementos da interseção entre os elementos do contexto atual e os elementos dos contextos anteriores.

O algoritmo mantém uma lista com o histórico dos escores brutos de anomalia. Os últimos elementos desta lista são comparados com um *threshold*. O escore final de anomalia será zero caso exista algum elemento no histórico de anomalias maior que o *threshold*. Caso contrário, o escore final irá corresponder ao escore bruto.

4.5.3 Earthgecko Skyline

Earthgecko Skyline [Wilson, 2019], [Earthgecko, 2019] é um projeto de código aberto, criado a partir do Etsy Skyline, com o objetivo de dar continuidade ao mesmo e desenvolver melhorias. O projeto Skyline, descrito na seção 4.6.6, foi arquivado em 2014 pela empresa que o desenvolveu. A descrição do funcionamento

[12] https://github.com/numenta/NAB/tree/master/nab/detectors/context_ose
[13] https://github.com/smirmik/CAD

do earthgecko Skyline foi feita através da análise do código fonte disponível no repositório do NAB[14].

Earthgecko Skyline utiliza um conjunto de testes estatísticos para detecção de anomalias e um esquema de votação para gerar a pontuação final de anomalia. Além disso, o algoritmo utiliza a técnica de janela deslizante, desta forma, os testes estatísticos são feitos apenas com as últimas observações analisadas.

As estatísticas utilizadas pelo algoritmo nos testes são os seguintes: desvio absoluto mediano, média da primeira hora, desvio padrão da média, desvio padrão da média móvel, subtração da média acumulada, mínimos quadrados e caixas de histograma. O teste consiste em comparar a estatística com um determinado *threshold*, específico para cada estatística.

Para cada observação X_t, o algoritmo executa os testes estatísticos listados acima. Cada teste retorna 1, indicando que é uma anomalia, ou 0, indicando que não é anomalia.

O algoritmo possui parâmetros que definem o esquema de votação utilizado no cálculo da pontuação final de anomalia, que pode ser por consenso ou por média. Quando o esquema definido é por consenso, um número mínimo de testes estatísticos, definido pelo parâmetro $CONSENSUS$, deve retornar 1 para que a observação seja considerada anomalia. Quando o esquema for por média, então o escore de anomalia calculado será a média dos valores retornados pelos testes estatísticos. Para o resultado mostrado no *benchmark* da tabela 4.2 o esquema utilizado foi por consenso e o valor adotado para o parâmetro $CONSENSUS$ foi 5.

O algoritmo também implementa a funcionalidade $EXPIRATION\_TIME$, na qual alertas de anomalia não serão gerados duas vezes dentro de $EXPIRATION\_TIME$.

4.6 Demais Algoritmos do Benchmark do NAB

Nesta seção descrevemos resumidamente os demais detectores incluídos no *benchmark* do NAB: KNN CAD, Relative Entropy, Random Cut Forest, Twitter ADVec, Windowed Gaussian, Etsy Skyline, Bayesian Changepoint e EXPoSE.

4.6.1 KNN CAD

O detector KNN CAD (*K-Nearest Neighbors Conformal Anomaly Detection*) [Burnaev & Ishimtsev, 2016] ficou em terceiro lugar na competição NAB de 2016 para detecção de anomalia em tempo real [Numenta, Inc., 2016].

[14]https://github.com/numenta/NAB/tree/master/nab/detectors/earthgecko_skyline

KNN CAD é uma aplicação do método *Inductive Conformal Anomaly Detection* [Laxhammar & Göran 2015] em séries temporais e utiliza como medida de não conformidade o escore de anomalia KNN.

4.6.2 Relative Entropy

O detector Relative Entropy [Wang et al. 2011] foi desenvolvido pela HP Labs[15].

O algoritmo é baseado em uma abordagem de teste de hipóteses que compara dados observados contra múltiplas hipóteses nulas, representando frequências de dados quantificados sobre uma janela. Se os dados observados não são vistos e não concordam com qualquer hipótese existente, ela é declarada anômala e uma nova hipótese é criada. Caso contrário, é declarado não anômalo, desde que a hipótese aceita ocorra com frequência suficiente. A decisão de aceitar/rejeitar uma hipótese nula baseia-se na entropia relativa comparada com um *threshold* de probabilidade de falso negativo aceitável determinado pela distribuição *chi-squared*[16].

4.6.3 Random Cut Forest

O detector Random Cut Forest foi desenvolvido pela AWS[17]. O código é proprietário, mas a descrição do algoritmo pode ser vista em [Guha et al. 2016]. Para avaliar/utilizar este detector é necessário criar uma aplicação na AWS do tipo Amazon Kinesis Data Analytics[18] utilizando o algoritmo RANDOM_CUT_FOREST[19].

O algoritmo utiliza uma técnica de detecção de anomalias baseada em cortes aleatórios de estruturas de dados em árvores.

4.6.4 Twitter ADVec

O Twitter[20] lançou em 2015 duas versões de um algoritmo de detecção de anomalias em tempo real [Kejariwal, 2015]. Ambas as versões são utilizadas em aplicações reais e o código fonte (em R) está disponível em [Twitter, Inc., 2015]. O algoritmo baseia-se no teste ESD (Extreme Studentized Deviate) Generalizado [Rosner, 1983] e é uma combinação de técnicas estatísticas para detecção de anomalias. Emprega a decomposição de séries temporais e usa métricas estatísticas ro-

[15]https://www.labs.hpe.com/
[16]https://en.wikipedia.org/wiki/Chi-squared_distribution
[17]https://aws.amazon.com
[18]https://aws.amazon.com/kinesis/dataanalytics/
[19]https://docs.aws.amazon.com/kinesisanalytics/latest/sqlref/sqlrfrandomcutforest.html
[20]https://twitter.com/

bustas, como a mediana junto com o ESD. Além disso, para séries temporais longas, como 6 meses de dados, o algoritmo emprega a aproximação por partes.

Uma versão do algoritmo, AnomalyDetectionVec, é mais genérica, detecta anomalias sem utilizar os registros de data e hora, mas requer ajuste manual da periodicidade. Uma segunda versão do algoritmo, AnomalyDetectionTs, explora registros de data e hora para detectar periodicidade e pode detectar anomalias de curto prazo (intra-dia) e de longo prazo (inter-dia). Infelizmente, AnomalyDetectionTs não pôde calcular os parâmetros de período necessários para alguns dos arquivos de dados do NAB, por este motivo os autores do NAB não o incluíram no *benchmark* da tabela 4.2.

4.6.5 Windowed Gaussian

Um detector de janela deslizante que calcula a pontuação de anomalia de um ponto de dados calculando sua probabilidade a partir da distribuição gaussiana sobre uma janela de pontos de dados anteriores.

O código fonte está disponível no repositório do NAB[21]

4.6.6 Etsy Skyline

Skyline [Etsy, Inc., 2013] é um sistema de detecção de anomalias em tempo real desenvolvido pela Etsy.com[22] para monitorar o tráfego do seu site. O projeto Skyline foi arquivado em 2014 pela empresa que o desenvolveu, mas a partir dele foi criado o projeto Earthgecko Skyline descrito na seção 4.5.3.

O core dos dois algoritmos é o mesmo: um conjunto de testes estatísticos de detecção de anomalias e um esquema de votação para gerar a pontuação final da anomalia.

Mas existem três diferenças principais entre eles. A primeira é que o esquema de votação utilizado pelo Etsy Skyline é o de média, ou seja, o escore final calculado é a média do que é retornado pelos testes estatísticos. A segunda diferença é que não foi implementada a funcionalidade $EXPIRATION\_TIME$, que não permite gerar alertas de anomalia duas vezes dentro do período $EXPIRATION\_TIME$. Esse é um recurso muito importante no Earthgecko Skyline para garantir um número menor de falsos positivos. Por fim, Etsy Skyline não utiliza a técnica de janela deslizante, dessa forma, os testes estatísticos são realizados no conjunto completo de observações analisadas.

[21] https://github.com/numenta/NAB/tree/master/nab/detectors/gaussian
[22] https://www.etsy.com

4.6.7
Bayesian Changepoint

O detector Bayesian Changepoint é a implementação do algoritmo Bayesian Online Changepoint Detection descrito em [Adams & MacKay 2007].

O algoritmo calcula, para cada registro na etapa x em um fluxo de dados, a probabilidade de que o registro atual seja parte de um fluxo de comprimento n para todo $n \leq x$. Para um determinado registro, se o máximo de todas as probabilidades corresponder a um comprimento de fluxo igual a zero, o registro representará um ponto de mudança (*Changepoint*) no fluxo de dados. Essas probabilidades são usadas para calcular escores de anomalia.

4.6.8
EXPoSE

O detector EXPoSE é a implementação do algoritmo descrito no artigo Expected Similarity Estimation for Lage-Scale Batch and Streaming Anomaly Detection, [Schneider et al., 2016].

O EXPoSE calcula a probabilidade de um ponto de dados ser normal usando o produto interno de seu mapa de recursos com incorporação de kernel de pontos de dados anteriores. Isso mede a similaridade de um ponto de dados com pontos anteriores sem assumir uma distribuição de dados subjacente.

4.7
Datasets Adicionais

Além do *dataset* incluído do NAB, utilizamos outros dois *datasets* para os testes realizados neste trabalho. Um deles criado e disponibilizado pelo Yahoo![23] e o outro criado especificamente para este trabalho com dados de produção de servidores da Globo.com[24].

4.7.1
Adicionando outros datasets ao NAB

Para adicionar um *dataset* externo para ser processado pelo *framework* NAB é necessário criar um arquivo com as janelas de anomalias para este *dataset*. Para gerar este arquivo, é necessário executar o *script combine_labels.py*[25] que recebe como entrada o diretório onde estão os arquivos com os dados das séries temporais (arquivos com registro de data e hora e um valor escalar) e um arquivo

[23] https://www.yahoo.com
[24] https://www.globo.com/
[25] https://github.com/numenta/NAB/blob/master/scripts/combine_labels.py

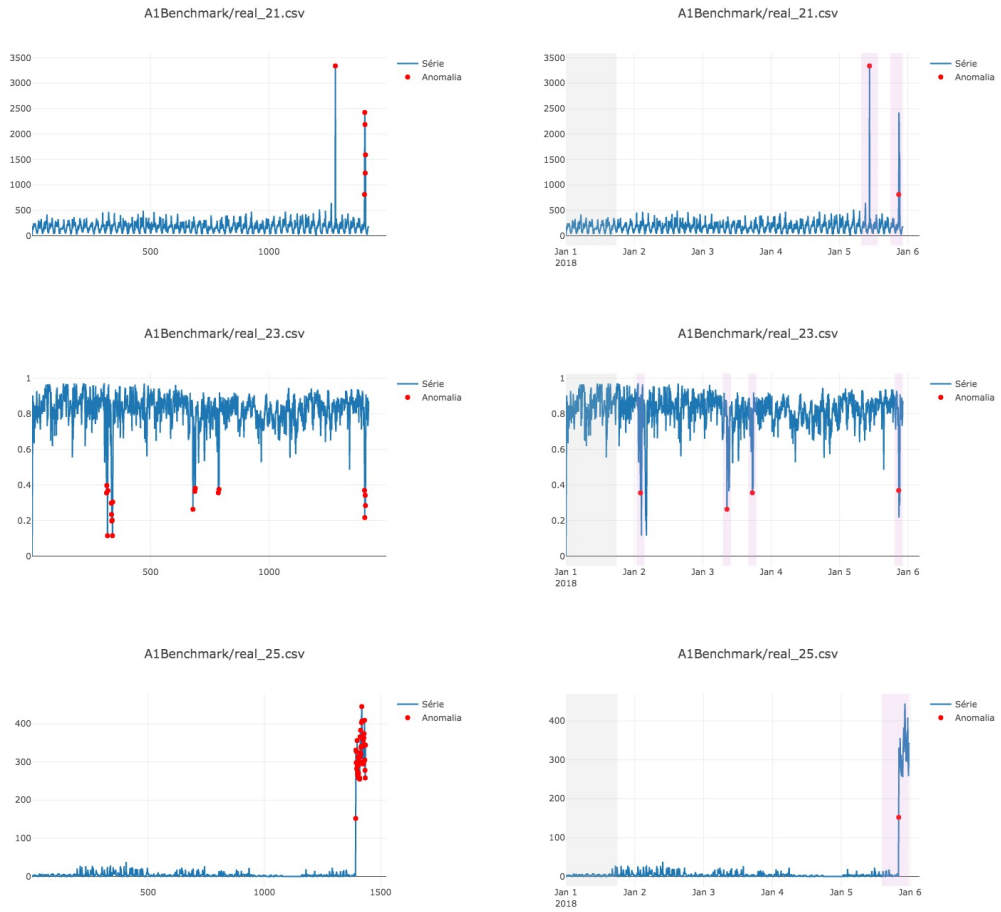
com os rótulos das anomalias (arquivo no formato json contendo as anotações das anomalias). Este procedimento está documentado em [Ahmad & Lavin, 2015b].

4.7.2 Dataset Yahoo! Webscope

O *dataset* do Yahoo! Webscope [Yahoo! Webscope, 2019] é composto por séries temporais juntamente com os rótulos de anomalias. Contém 4 conjuntos de dados: A1Benchmark, A2Benchmark, A3Benchmark e A4Benchmark. O primeiro foi criado a partir de dados reais de produção de alguns serviços do Yahoo!, os demais foram criados a partir de dados sintéticos. Neste trabalho iremos utilizar apenas o conjunto A1Benchmark que é o único contendo dados reais.

A1Benchmark é composto por 67 séries temporais, cada uma contendo entre 741 e 1.461 observações, totalizando 94.866 observações. Cada série é armazenada em um arquivo e cada linha contém um *timestamp*, um valor escalar e um booleano indicando se o registro é anômalo ou não. Os rótulos de anomalia incluídos nos arquivos foram definidos por humanos.

Os rótulos de anomalias que foram criados no *dataset* Yahoo! A1Benchmark possuem anomalias muito próximas umas das outras e para que não houvesse sobreposição das janelas de anomalia ao executar o *script combine_labels.py*, cujo procedimento foi documentado na seção 4.7.1, geramos novos rótulos de anomalia, utilizando a seguinte regra: a cada conjunto de anomalias próximas, pertencentes a uma mesma janela de anomalia, consideramos apenas a primeira anomalia e ignoramos as anomalias seguintes. A figura 4.7a mostra alguns exemplos de séries temporais do A1Benchmark e os rótulos de anomalias presentes nos arquivos originais, representados na figura pelos pontos em vermelho. A figura 4.7b mostra estas mesmas séries temporais, com apenas a primeira anomalia de cada janela de anomalia. A região sombreada da figura na cor cinza representa o período probatório, a região sombreada na cor roxa representa as janelas de anomalia e os pontos em vermelho as anomalias.



(a) Dataset A1Benchmark original

(b) Dataset A1Benchmark adaptado para o framework NAB

Figura 4.7: Exemplos de séries do dataset Yahoo! A1Benchmark

4.7.3 Globo.com-dataset

Neste trabalho criamos um *dataset* a partir de métricas de CPU de máquinas de produção do data center da Globo.com. O *dataset* é composto por 142 séries temporais univariadas, armazenada em arquivos, com registro de data e hora e um valor escalar. Cada série possui entre 3566 e 7488 observações, totalizando 1.012.150 observações. Os rótulos de anomalia foram definidos por humano a partir da observação gráfica da série e algumas séries não possuem anomalias. Os rótulos de anomalia foram salvos num arquivo json. A escolha das séries temporais que compõem este *dataset* foi feita de forma aleatória.

A figura 4.8 ilustra alguns exemplos de séries contidas no *dataset* da Globo.com. A região sombreada da figura na cor cinza representa o período probatório, a região sombreada na cor roxa representa as janelas de anomalia e os

pontos em vermelho as anomalias. As duas primeiras séries da figura não contêm anomalias.

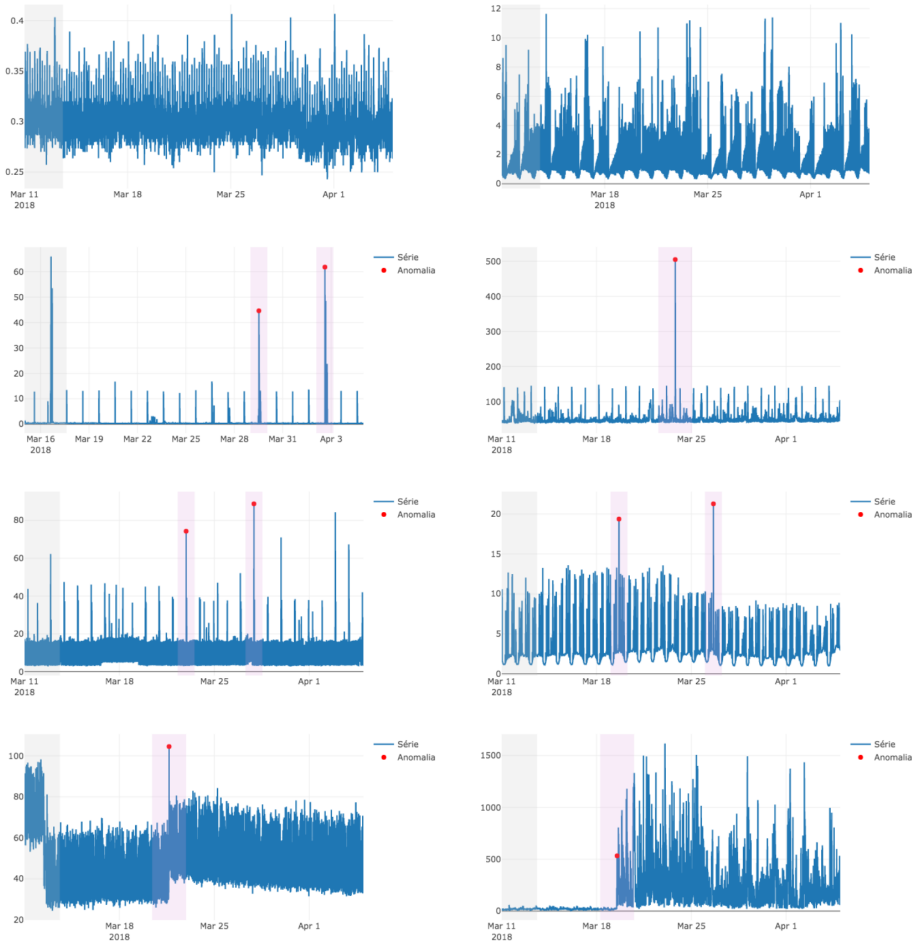


Figura 4.8: Exemplos de séries do dataset Globo.com

5
Algoritmos Propostos

Este capítulo descreve os algoritmos propostos e implementados neste trabalho. Na primeira seção descrevemos um algoritmo base que é utilizado para instanciar e avaliar diferentes tipos de algoritmos de detecção. Em seguida, os algoritmos de detecção desenvolvidos são detalhados. Apresentamos quatro algoritmos baseados em análises estatísticas clássicas, utilizadas em diferentes pesquisas que propõem técnicas de detecção de anomalias. Posteriormente, descrevemos o detector de anomalias Ensemble Detector que utiliza a técnica de treinamento supervisionado. Por fim, apresentamos o algoritmo proposto que chamamos de DASRS. Descrevemos em detalhes como o DASRS guarda os padrões de sequências de dados das séries temporais analisadas e utiliza esta informação para identificar novos padrões e gerar um escore de anomalia.

5.1
Algoritmo Base

O algoritmo 1 mostra um exemplo básico de como os detectores podem ser chamados. Este algoritmo básico é responsável por instanciar o detector (qualquer um dos detectores deste trabalho), ler as observações x_i da série temporal X e chamar o método $handleRecord$ do detector, passando a observação x_i como argumento e recebendo como resposta um escore de anomalia. Alguns detectores deste trabalho possuem parâmetros opcionais (se não for informado, utiliza valores *default*). Estes parâmetros estão representados no algoritmo 1 por $arg1, arg2, \ldots$ e são passados na hora de instanciar o detector. Cada detector tem um $threshold$ que indica se o escore retornado é uma anomalia ou não. O que será feito com o resultado do escore gerado depende da aplicação. Os métodos $saveAnomalyScore$ e $displayAlarm$ deverão ser implementados pela aplicação. $saveAnomalyScore$, por exemplo, pode ser salvar em um banco de dados e $displayAlarm$ pode ser exibir em um painel um alarme indicando que uma anomalia foi detectada.

Algoritmo 1: Exemplo básico de como os detectores são chamados e como os resultados podem ser utilizados

Entrada:

- X: sequência de observações $(x_1, x_2, x_3, \dots)$
- $Detector$: classe do detector de anomalia
- $threshold$: limite que identifica o que é anomalia a partir do escore gerado pelo detector

```
1 d ← Detector()
2 d.init(arg1, arg2, ...)
3 para cada observação x_i em X faça
4     anomalyScore ← d.handleRecord(x_i)
5     saveAnomalyScore(anomalyScore)
6     se anomalyScore >= threshold então
7         displayAlarm()
```

5.2 Algoritmos Baseados em Testes Estatísticos

Three-sigma rule, *Median Absolute Deviation*, *Modified three-sigma rule* e *Tukey's method* são análises estatísticas clássicas utilizadas para detecção de anomalias.

Em todos os algoritmos implementados nesta seção a análise estatística é realizada em um subconjunto da série temporal, utilizando janelas deslizantes. A janela deslizante é uma estrutura de dados de tamanho constante em que a adição de novos dados ocorre junto com a exclusão do dado mais antigo. Nos algoritmos descritos nessa sessão, a janela deslizante tem tamanho configurável (variável $windowSize$) e deslocamento de tamanho 1, ou seja, a cada nova observação a ser analisada pelo detector, a janela é deslocada em 1 (uma) unidade. A função $append(x_i)$ é usada para adicionar a observação x_i a janela e a função $pop(0)$ para remover o primeiro elemento da janela. A utilização da janela é necessária para limitar o conjunto de dados que é armazenado pelo detector e também submetido à análise estatística. Sem essa limitação, a memória utilizada pelo detector e o tempo para realizar o teste crescem indefinidamente. Nesse tipo de abordagem as informações consideradas antigas não interferem no resultado atual da análise.

Muitos dos algoritmos estudados, como earthgecko Skyline, KNN CAD, Relative Entropy e Windowed Gaussian utilizam a técnica de janela deslizante. Porém, somente o algoritmo Windowed Gaussian mantém armazenado apenas as observações utilizadas na análise. Os demais algoritmos mantêm armazenado todo o histórico de observações, apesar de utilizar na análise apenas as observações que estão na janela.

Anomalias muito próximas geralmente estão associadas ao mesmo fenômeno

(ou causa raiz). Por isso, apenas um alarme é necessário para que a anomalia seja investigada. Para evitar que os algoritmos detectem anomalias muito próximas umas das outras, acrescentamos uma variável que controla por quanto tempo o algoritmo deve ficar sem detectar novas anomalias após ter identificado uma. Com esta técnica também conseguimos diminuir a quantidade de falsos positivos nos *datasets* utilizados nos testes. Nos algoritmos descritos nesta seção, o período que o detector fica sem detectar novas anomalias é controlado pela variável $restPeriod$.

Vários dos algoritmos estudados utilizam alguma técnica para diminuir a quantidade de falsos positivos. CAD OSE, earthgecko Skyline e KNN CAD implementam técnicas semelhantes a descrita acima. Enquanto o Numenta HTM, a partir do escore bruto de anomalia calculado pelo algoritmo, utiliza um método que calcula a probabilidade deste escore bruto ser uma anomalia.

5.2.1 Three-sigma rule

Three-sigma rule faz parte de uma regra da estatística, chamada regra empírica [Ross, 2004]. Para um conjunto de dados com distribuição aproximadamente normal, com média (μ) e desvio padrão (σ), a regra empírica afirma que:

- Aproximadamente, 68% dos dados estão dentro de 1 σ padrão da μ;
- Aproximadamente, 95% dos dados se encontram dentro de 2 σ da μ;
- Aproximadamente, 99,7% dos dados estão dentro de 3 σ da μ.

A regra empírica, representada na figura 5.1, também é chamada de regra 68-95-99,7.

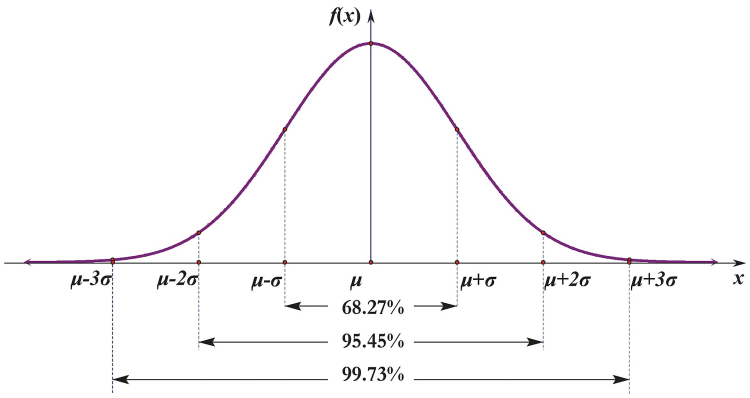


Figura 5.1: Curva normal e os percentuais de distribuição calculados pela regra empírica

Three-sigma rule corresponde a terceira afirmação da regra empírica, que diz que quase todos os valores (99,73%) encontram-se dentro de 3 desvios padrão (σ) da média (μ). A regra *three-sigma* é um critério bastante conhecido e amplamente

utilizado para detecção de anomalia, alguns autores até referem-se à regra *three-sigma* para a definição de anomalia [Hekimoglu & Koch, 2000].

Na equação 5-1, se x_i estiver fora deste intervalo, é bem provável que se trate de uma anomalia.

$$|x_i - \mu| > 3\sigma \tag{5-1}$$

No algoritmo 2, mostramos como a equação 5-1 é utilizada para identificação das anomalias. No algoritmo também mostramos a utilização das janelas deslizantes no cálculo de μ e σ e também como a variável $restPeriod$ é utilizada para não gerar anomalias muito próximas.

Algoritmo 2: Three Sigma Rule Anomaly Detector

Entrada:

- $windowSize$: tamanho da janela deslizante
- $restPeriod$: período sem detectar anomalia após ter detectado uma
- x_i: observação corrente

Saída: $AnomalyScore \in$ [0,1]: escore da anomalia

Function `init` $(windowSize, restPeriod)$**:**
    $buffer \leftarrow []$
    $rest \leftarrow -1$

Function `threeSigmaOutlier` $(buffer, x_i)$**:**
    $\mu \leftarrow Media(buffer)$
    $\sigma \leftarrow DesvioPadrao(buffer)$
    se $|x_i - \mu| > 3\sigma$ **então**
        retorna 1
    senão
        retorna 0

Function `handleRecord` (x_i)**:**
    $buffer.append(x_i)$
    se $len(buffer) < windowSize$ **então**
        retorna 0
    $anomalyScore \leftarrow threeSigmaOutlier(buffer, x_i)$
    $buffer.pop(0)$
    se $rest > 0$ **então**
        $rest \leftarrow rest - 1$
        retorna 0
    se $anomalyScore == 1$ **então**
        $rest \leftarrow restPeriod$
    retorna $anomalyScore$

5.2.2

Median Absolute Deviation

A média e o desvio padrão podem ser afetadas por uma ou mais observações com valor extremo. Para evitar este problema, [Leys et al. 2013] sugere utilizar desvio absoluto da mediana (MAD - *median absolute deviation*) no lugar da média e do desvio padrão.

A fórmula para obter o valor do MAD é dada pela equação abaixo:

$$MAD = mediana(|x_i - \widetilde{x}|) \tag{5-2}$$

onde x_i é a observação analisada e $\widetilde{x}$ é a mediana da amostra de dados.

[Miller, 1991] sugere utilizar um *threshold* de valor 3 para identificação das anomalias: $MAD > 3$.

O algoritmo 3 mostra como as equações acima são utilizadas na identificação das anomalias. As janelas deslizantes e a variável $restPeriod$ são utilizados da mesma forma que no algoritmo 2.

Algoritmo 3: Median Absolute Deviation Anomaly Detector

Entrada:

- $windowSize$: tamanho da janela deslizante
- $restPeriod$: período sem detectar anomalia após ter detectado uma
- x_i: observação corrente

Saída: $AnomalyScore \in$ [0,1]: escore da anomalia

```
Function init(windowSize, restPeriod):
    buffer ← []
    rest ← −1

Function mad(buffer, x_i):
    x̃ ← Mediana(buffer)
    MAD ← Mediana(|x_i − x̃|)
    se MAD > 3 então
        retorna 1
    senão
        retorna 0

Function handleRecord(x_i):
    buffer.append(x_i)
    se len(buffer) < windowSize então
        retorna 0
    anomalyScore ← mad(buffer, x_i)
    buffer.pop(0)
    se rest > 0 então
        rest ← rest − 1
        retorna 0
    se anomalyScore == 1 então
        rest ← restPeriod
    retorna anomalyScore
```


5.2.3 Modified three-sigma

[Iglewicz & Hoaglin, 1993] também desenvolveram um método que é menos sensível a dados com valor extremo dos que utilizam média e o desvio padrão. Este método utiliza mediana e o desvio absoluto da mediana (MAD - *median absolute deviation*. O cálculo do MAD e Modified three-sigma (M_σ) é feito através das equações abaixo:

$$MAD = mediana(|x_i - \widetilde{x}|) \tag{5-3}$$

$$M_\sigma = \frac{0.6745 \times (x_i - \widetilde{x})}{MAD} \tag{5-4}$$

onde x_i é a observação analisada e $\widetilde{x}$ é a mediana da amostra de dados.

[Iglewicz & Hoaglin, 1993] sugere que observações sejam rotuladas como anomalias quando $|M_\sigma| > 3.5$.

O algoritmo 4 mostra como as equações acima são utilizadas na identificação das anomalias. As janelas deslizantes e a variável $restPeriod$ são utilizados da mesma forma que no algoritmo 2.

Algoritmo 4: Modified Three Sigma Rule Anomaly Detector

Entrada:

- $windowSize$: tamanho da janela deslizante
- $restPeriod$: período sem detectar anomalia após ter detectado uma
- x_i: observação corrente

Saída: $AnomalyScore \in$ [0,1]: escore da anomalia

```
1  Function init(windowSize, restPeriod):
2      buffer ← []
3      rest ← −1

4  Function modifiedthreeSigmaOutlier(buffer, x_i):
5      x̃ ← Mediana(buffer)
6      MAD ← Mediana(|x_i − x̃|)
7      M_σ ← 0.6745 ∗ (x_i − x̃)/MAD
8      se |M_σ| > 3.5 então
9          retorna 1
10     senão
11         retorna 0

12 Function handleRecord(x_i):
13     buffer.append(x_i)
14     se len(buffer) < windowSize então
15         retorna 0
16     anomalyScore ← modifiedthreeSigmaOutlier(buffer, x_i)
17     buffer.pop(0)
18     se rest > 0 então
19         rest ← rest − 1
20         retorna 0
21     se anomalyScore == 1 então
22         rest ← restPeriod
23     retorna anomalyScore
```

5.2.4 Tukey's method

John Tukey [Tukey, 1977] desenvolveu um método simples e eficiente para detecção de anomalias. O método leva em consideração o cálculo de quartis, o que faz com que seja menos sensível a dados com valores extremos que métodos que usam média e desvio padrão. Quartis são valores que dividem um conjunto

ordenado de dados em quatro partes iguais, e assim cada parte representa 1/4 da amostra. Há, portanto, três quartis: Q1, Q2 e Q3.

- Q1: é chamado de primeiro quartil, ou seja, valor que deixa 25% dos elementos à sua esquerda e 75% dos elementos à sua direita;
- Q2: é chamado de segundo quartil e coincide com a mediana, ou seja, 50% dos elementos estão à sua esquerda e 50% à sua direita;
- Q3: é chamado de terceiro quartil, ou seja, valor que deixa 75% dos elementos à sua esquerda e 25% à sua direita.

A diferença entre o terceiro e primeiro quartis é chamado intervalo interquartil (IQR - *interquartile range*):

$$IQR = Q3 - Q1 \tag{5-5}$$

A partir dos valores dos quartis e do IQR, Tukey definiu os limites inferior e superior em que os dados são considerados normais. Os limites inferior e superior são definidos respectivamente pelas equações abaixo:

$$LowerBound = Q1 - 1.5 \times IQR \tag{5-6}$$

$$UpperBound = Q3 + 1.5 \times IQR \tag{5-7}$$

Observações menores que $LowerBound$ ou maiores que $UpperBound$ são consideradas possíveis anomalias.

O algoritmo 5 mostra como o método foi implementado. As janelas deslizantes e a variável $restPeriod$ são utilizados da mesma forma que no algoritmo 2.

Algoritmo 5: Tukey's method Anomaly Detector

Entrada:

- $windowSize$: tamanho da janela deslizante
- $restPeriod$: período sem detectar anomalia após ter detectado uma
- x_i: observação corrente

Saída: $AnomalyScore \in$ [0,1]: escore da anomalia

Function `init` $(windowSize, restPeriod)$ **:**
    $buffer \leftarrow []$
    $rest \leftarrow -1$

Function `TukeyOutlier` $(buffer, x_i)$ **:**
    $Q1 \leftarrow Quartil(buffer, 1)$
    $Q3 \leftarrow Quartil(buffer, 3)$
    $iqr \leftarrow Q3 - Q1$
    $lowerBound \leftarrow Q1 - (iqr * 1.5)$
    $upperBound \leftarrow Q3 + (iqr * 1.5)$
    se $(x_i > upperBound)$ ***OU*** $(x_i < lowerBound)$ **então**
        retorna 1
    senão
        retorna 0

Function `handleRecord` (x_i) **:**
    $buffer.append(x_i)$
    se $len(buffer) < windowSize$ **então**
        retorna 0
    $anomalyScore \leftarrow TukeyOutlier(buffer, x_i)$
    $buffer.pop(0)$
    se $rest > 0$ **então**
        $rest \leftarrow rest - 1$
        retorna 0
    se $anomalyScore == 1$ **então**
        $rest \leftarrow restPeriod$
    retorna $anomalyScore$

5.3 Ensemble Detector

O detector apresentado nessa seção combina os resultados dos algoritmos de detecção de anomalias do estado da arte para fornecer resultados mais precisos do que em cada algoritmo individual. Nesta abordagem, dividimos o processo em duas etapas. A primeira, chamada etapa de treinamento supervisionado, consiste em criar um conjunto de classes de séries temporais e um modelo de classificação, em seguida, classificar um conjunto de séries temporais e verificar qual detector tem melhor resultado para cada uma das classes. O modelo de classificação e a informação do melhor detector de cada classe são utilizados como entrada para a

segunda etapa, chamada etapa de detecção de anomalias. A segunda etapa consiste em executar o detector com melhor resultado de acordo com a classe da série analisada.

5.3.1 Etapa de Treinamento Supervisionado

A figura 5.2 mostra com mais detalhes como funciona a primeira etapa. Na figura, os dados de entrada estão representados com a cor amarela e os dados de saída, que serão utilizados na etapa seguinte, estão representados com a cor verde. Os dados de entrada são: um conjunto de séries temporais, seus rótulos de anomalia e um conjunto de detectores. Um conjunto de C classes é criado a partir das séries temporais e estas classes, juntamente com as séries temporais, são processadas por um classificador para criar um modelo de classificação. A partir do modelo de classificação as séries são novamente divididas em C subconjuntos (classes). As séries temporais são analisadas pelos diferentes detectores e seus resultados são comparados com os rótulos de anomalias. Assim, calculamos a pontuação de cada detector, em cada uma das classes, em função de sua assertividade. Portanto, um dicionário com o detector que obteve o melhor resultado em cada classe é gerado.

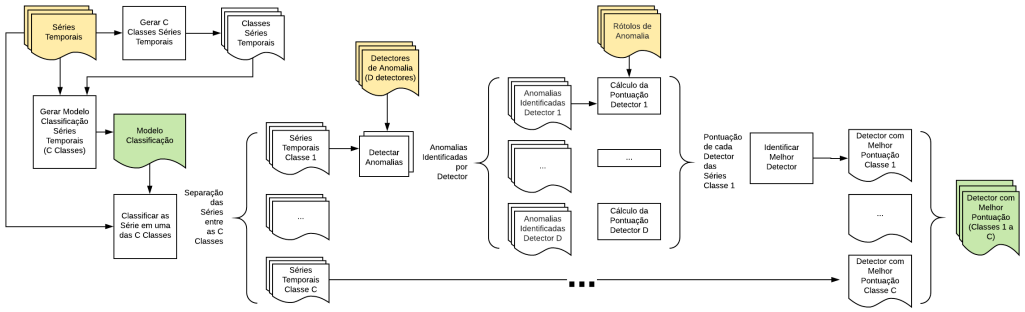


Figura 5.2: Etapa de treinamento supervisionado

Nesta etapa utilizamos alguns componentes do *framework* NAB. Utilizamos as séries temporais com os rótulos de anomalia relacionados ao *dataset* e o conjunto de detectores previamente incluídos no *framework*, que são utilizados pelos algoritmos de detecção do *benchmark*, além da lógica de pontuação nativa do NAB. Para criar as classes utilizamos o componente KShape da biblioteca scikit-learn[1]. KShape é um algoritmo de clusterização de séries temporais baseado na forma (*shape*) da série. A quantidade de clusteres (classes) criados é definido por parâmetro. E na criação do modelo de classificação das séries utilizamos o classificador DecisionTreeClassifier da mesma biblioteca scikit-learn.

[1]https://scikit-learn.org/stable/

Os detalhes de nossa implementação estão descritos no algoritmo 6.

Algoritmo 6: Ensemble Detector - Etapa de Treinamento

Entrada:

- X_S: S sequências de observações $(x_1, x_2, x_3, \dots)_1, \dots, (x_1, x_2, x_3, \dots)_S$
- R: Rótulos de anomalia de todas as séries de entrada
- D: Lista de detectores $(d_1, d_2, \dots, d_D)$

Saída:

- $BestDetectorPerClass$: Dicionário com a lista de classes e o melhor detector de cada classe
- $ClassifierModel$: Modelo de classificação de séries temporais

Function `createClasses(`X_S`)`**:**
  $classes \leftarrow Clustering(numberClusters)$
  $classes.fit(X_S)$
  retorna $classes$

Function `createClassifierModel(`X_S`)`**:**
  $model \leftarrow Classifier()$
  $model.fit(X_S)$
  retorna $model$

Function `splitSeries(`$model, X_S$`)`**:**
  $classes \leftarrow dict()$
  para cada *sequence* S *em* X_S **faça**
    $class = model.predict(S)$
    se $class \in classes$ **então**
      $classes[class].append(S)$
    senão
      $classes[class] \leftarrow []$
      $classes[class].append(S)$
  retorna $classes$

$classes \leftarrow createClasses(X_S)$
$model \leftarrow createClassifierModel(X_S, classes)$
$SaveModel(model)$
$classes \leftarrow splitSeries(model, X_S)$
para cada *class* C *em* $classes$ **faça**
  para cada *sequence* S *em* C **faça**
    para cada *Detector em* D **faça**
      $d \leftarrow Detector()$
      $d.init()$
      para cada *observação* x_i *em* S **faça**
        $anomalyScore \leftarrow d.handleRecord(x_i)$
        $saveAnomalyScore(anomalyScore)$
  $nabScores \leftarrow []$
  para cada *detector em* D **faça**
    $score \leftarrow$
     $computeNabScore(sequence, R, detector, anomalyScoreDetector)$
    $nabScores.append(score)$
  $bestDetectorClass \leftarrow getBestDetector(class, nabScores, d)$

5.3.2
Etapa de Detecção de Anomalias

A figura 5.3 mostra como é executada a segunda etapa. Ao analisar os dados de uma série temporal a primeira ação é identificar qual classe pertence a série. Isso é realizado utilizando-se o modelo de classificação que foi gerado na etapa anterior. Além disso, o conjunto de detectores é o mesmo utilizado na etapa anterior. Depois de realizar a classificação da série e da criação de um dicionário que informa qual é o melhor detector para cada classe, seleciona-se qual detector será utilizado na análise da série para identificar anomalias. Uma vez identificado o detector, o algoritmo Ensemble Detector instancia o detector escolhido e lhe envia as observações x_i da série temporal. O detector selecionado, então, gera um escore de anomalia para cada observação analisada.

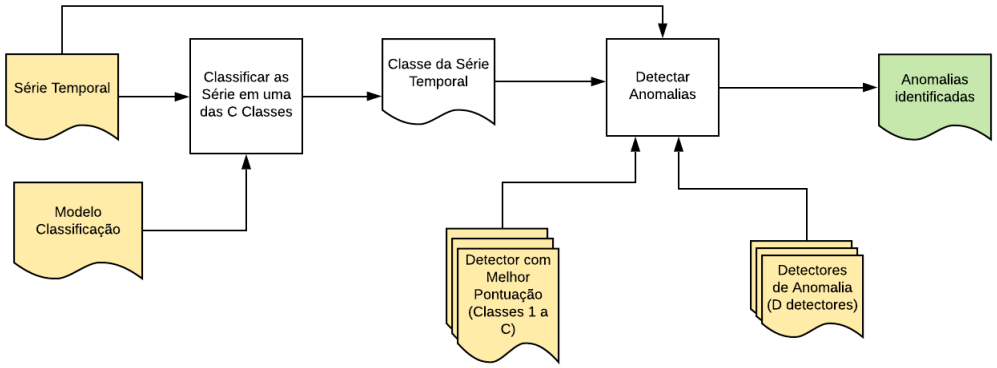


Figura 5.3: Etapa de detecção de anomalias

Os detalhes da implementação desta etapa são mostrados no algoritmo 7.

Algoritmo 7: Ensemble Detector - Etapa de Detecção de Anomalias

Entrada:

- X: sequência de observações $(x_1, x_2, x_3, \dots)$
- $model$: Modelo de classificação
- D: Lista de detectores $(d_1, d_2, \dots, d_D)$
- $BestDetectorPerClass$: Dicionário com a lista de classes e o melhor detector de cada classe
- $ClassifierModel$: Modelo de classificação de séries temporais

Saída: $AnomalyScore \in$ [0,1]: escore da anomalia

```
class = model.predict(X)
Detector ← BestDetectorPerClass.getDetector(class)
d ← Detector()
d.init()
para cada observação x_i em X faça
    anomalyScore ← d.handleRecord(x_i)
    saveAnomalyScore(anomalyScore)
```

O Ensemble Detector possui algumas fraquezas quando é utilizado no processamento *streaming*. Para escolher o detector mais adequado para analisar uma série de dados o Ensemble Detector precisa primeiro identificar a que classe a série pertence. Por isso, o algoritmo armazena uma certa quantidade de observações antes que seu modelo de classificação seja capaz de categorizar a série analisada. Isso é um problema quando o algoritmo é utilizado no processamento *streaming* porque não é possível gerar resultados em tempo real, antes da classificação da série analisada. Uma vez identificado o melhor detector as observações armazenadas em memória são enviadas para o detector selecionado e em seguida processadas pelo algoritmo que realiza a análise. Além de impossibilitar a geração de resultados em tempo real as características descritas contribuem para o aumento da complexidade do Ensemble Detector. Outra questão que precisa ser aperfeiçoada está relacionada a mudança de conceito (descrito na seção 2.3) que pode ocorrer em uma série temporal. Isso porque após uma mudança de conceito uma série pode passar a pertencer a uma classe diferente da que foi definida antes da mudança. Por fim, a qualidade do modelo de classificação da série temporal depende do tamanho e qualidade do *dataset* utilizado pelo classificador. Entretanto, é tarefa difícil encontrar pronto ou criar um *dataset* com dados rotulados que represente todos os tipos de comportamento.

5.4 DASRS

A ideia básica do algoritmo proposto DASRS é identificar e contabilizar as sequências de valores normalizados que aparecem numa série temporal e gerar um escore em função da quantidade de vezes que cada sequência foi identificada. Na

primeira vez que uma determinada sequência é identificada o escore retornado é o maior possível, pois o algoritmo interpreta como um comportamento novo. À medida que aumenta a quantidade de vezes que a sequência é encontrada, o escore retornado fica cada vez menor.

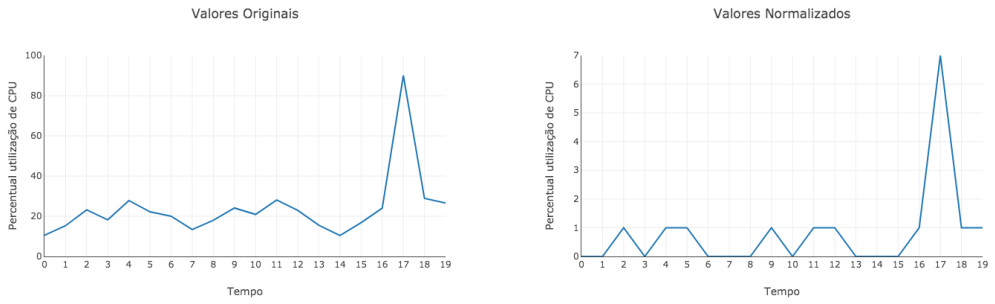
Sendo X_t a série temporal formada pelas observações $x_1, x_2, \ldots$, uma sequência de X_t é um subconjunto de X_t formado por elementos consecutivos, por exemplo, $x_i, \ldots, x_j$, com $i < j$.

A normalização aplicada pelo algoritmo consiste em transformar o valor da observação em um número inteiro entre 0 e um fator de normalização que iremos chamar de θ. A equação abaixo representa as operações realizadas em x_i para obter seu valor normalizado (x'_i):

$$x'_i = \left\lfloor \frac{x_i - min_X}{max_X - min_X} \times \theta \right\rfloor \tag{5-8}$$

onde x_i é a observação de entrada ($x_i \in \mathbb{R}$), min_X e max_X são respectivamente o menor e maior valor de observação possível da série temporal X_t, θ representa o fator de normalização, x'_i é o valor normalizado da observação x_i, $x'_i \in \mathbb{N}$ e $0 \leq x'_i \leq \theta$.

Ao normalizar os valores observados é possível limitar a quantidade de sequências distintas sem alterar as características da série temporal. A figura 5.4a representa uma série temporal com observações de percentual de utilização de CPU. Originalmente o valor mínimo do percentual de utilização de CPU é 0 e o valor máximo é 100. Entretanto, é possível utilizar um θ com valor 7 para criar a série temporal com dados normalizados ilustrada na figura 5.4b. Depois da normalização os valores observados passam a variar entre 0 e 7. Contudo é possível observar nos gráficos 5.4a e 5.4b que a normalização limitou a quantidade de sequências distintas mantendo as características da série temporal.



(a) Série temporal com os valores originais (b) Série temporal com os valores normalizados

Figura 5.4: Exemplo de série temporal e a normalização aplicada

Ao calcular o escore de anomalia levamos em conta a sequência normalizada corrente e a quantidade de vezes que essa sequência ocorreu no passado. A

quantidade de elementos da sequência é definida por um parâmetro, que decidimos chamar de $sequenceSize$. A sequência normalizada corrente é formada pelos últimos $sequenceSize$ elementos normalizados da série temporal. Guardamos numa estrutura de dados a quantidade de vezes que cada sequência apareceu durante o processamento da série temporal. A equação abaixo é utilizada no cálculo do escore de anomalia:

$$score = \frac{1}{occurrences} \tag{5-9}$$

onde $occurrences$ representa a quantidade de vezes que a sequência corrente foi encontrada.

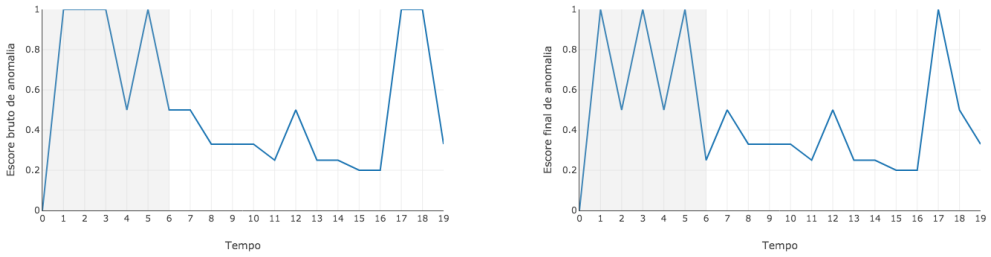
É esperado que no início do processamento de uma série temporal o algoritmo gere escores altos de anomalia. Isso ocorre porque até então as sequências foram encontradas poucas vezes pelo algoritmo. Por este motivo, deve-se considerar o tempo inicial de processamento de uma série como período de treinamento, ou período probatório, e as anomalias identificadas neste período não devem ser consideradas para disparar alarmes.

O escore gerado pela equação 5-9 não é o escore final. Conforme explicado em [Ahmad et al., 2017], muitas séries de dados possuem comportamento imprevisível, causado por ruídos ou pela natureza aleatória de algumas métricas, gerando um grande número de falsos positivos. Para lidar com isso, desenvolvemos duas versões do DASRS. A primeira, DASRS Rest, define um período, após identificar uma anomalia, em que o escore final de anomalia deverá ser suavizado. A segunda versão, DASRS Likelihood, utiliza a métrica *anomaly likelihood*, descrita na seção 4.5.1.2.

5.4.1 DASRS Rest

No DASRS, as anomalias muito próximas são tratadas como associadas ao mesmo fenômeno. Por isso, após identificar uma anomalia, os escores seguintes tem seu valor diminuído, por um período definido pelo parâmetro $restPeriod$. A função $getRestScore$ do algoritmo 8 mostra como o parâmetro $restPeriod$ é utilizado para atenuar o score final de anomalia.

As figuras 5.5a e 5.5b mostram respectivamente os escores bruto e final de anomalia gerados pelo DASRS para a série temporal da figura 5.4a, com $\theta = 7$, $sequenceSize = 2$ e $restPeriod = 2$. A área em cinza da figura representa o período de treinamento, período em que o algoritmo está identificando os padrões existente na série.



(a) Escore bruto de anomalia (b) Escore final de anomalia - Rest

Figura 5.5: Escores de anomalia calculados pelo DASRS Rest

A tabela 5.1 mostra o passo a passo de execução do DASRS para a série da figura 5.4a. A série temporal é representada pelas colunas Tempo e X, que informam o valor escalar de uma série temporal ao longo do tempo. X' é o valor normalizado de X, Sequência são os últimos 2 ($sequenceSize$) elementos de X', Ocorrências é a quantidade de vezes que a Sequência foi encontrada, Escore Bruto é o escore bruto de anomalia calculado pelo algoritmo e Escore Final é o escore retornado pelo algoritmo após atenuar o valor do escore bruto pelo período $restPeriod$.

Tabela 5.1: Passo a passo de processamento do DASRS Rest

Tempo	X	X'	Sequência	Ocorrências	Escore Bruto	Escore Final - Rest
0	10,5	0	-	-	0	0
1	15,3	0	(0, 0)	1	1	1
2	23,2	1	(0, 1)	1	1	0,5
3	18,2	0	(1, 0)	1	1	1
4	27,8	1	(0, 1)	2	0,5	0,5
5	22,2	1	(1, 1)	1	1	1
6	20,0	0	(1, 0)	2	0,5	0,25
7	13,4	0	(0, 0)	2	0,5	0,5
8	19,0	0	(0, 0)	3	0,33	0,33
9	24,1	1	(0, 1)	3	0,33	0,33
10	20,9	0	(1, 0)	3	0,33	0,33
11	28,1	1	(0, 1)	4	0,25	0,25
12	22,9	1	(1, 1)	2	0,5	0,5
13	15,5	0	(1, 0)	4	0,25	0,25
14	10,4	0	(0, 0)	4	0,25	0,25
15	16,8	0	(0, 0)	5	0,2	0,2
16	24,0	1	(0, 1)	5	0,2	0,2
17	90,0	7	(1, 7)	1	1	1
18	28,9	1	(7, 1)	1	1	0,5
19	26,6	1	(1, 1)	3	0,33	0,33

5.4.1.1 Escolha dos Parâmetros para o DASRS Rest

Para escolher os parâmetros do DASRS Rest realizamos uma bateria de testes variando os valores dos parâmetros de entrada. Diferentes combinações de parâmetros foram utilizadas enquanto nosso algoritmo foi executado para processar o *dataset* NAB. Assim, calculamos e registramos as pontuações obtidas com o NAB.

Os seguintes valores de parâmetros foram utilizados:

- θ: $2 \leq \theta \leq 10$;
- $sequenceSize$: $2 \leq sequenceSize \leq 5$;
- $restPeriod$: 0, 1, 2, $probationaryPeriod$, $probationaryPeriod/2$ e $probationaryPeriod/5$.

Onde θ é fator de normalização aplicado aos valores escalares da série temporal, $sequenceSize$ é a quantidade de elementos normalizados mantido pelo algoritmo, $restPeriod$ é o período após ter detectado uma anomalia que o algoritmo fica sem detectar novas anomalias (diminuindo o escore final de anomalia), $probationaryPeriod$ é o período probatório da série temporal calculado pelo NAB e corresponde a 0,15% do tamanho da série. Quando $restPeriod$ for 0 o escore retornado é o escore bruto calculado pelo algoritmo.

Além dos parâmetros acima, também variamos a forma como $restPeriod$ foi utilizada. Para isso, utilizamos dois métodos $Rest$ que iremos chamar de A e B, definidos abaixo:

- **Método Rest A**: após detectar uma anomalia, fica sem detectar anomalia pelo período correspondente a $restPeriod$;
- **Método Rest B**: mantém o histórico dos últimos escores brutos de anomalia. Se algum elemento desse histórico atingir o *threshold* de anomalia, o escore final retornado será baixo. A quantidade de elementos mantido nesse histórico corresponde ao valor de $restPeriod$.

As 50 maiores pontuações obtidas pelo algoritmo DASRS Rest utilizando o *dataset* NAB e diferentes combinações de parâmetros estão ilustrados na tabela 5.2. Quando realizamos os mesmos testes utilizando os *datasets* do Yahoo-A1Benchmark e Globo.com-dataset a combinação de parâmetros de melhor resultado também foi a descrita na linha 4 da tabela 5.2. Nas primeiras três combinações utilizadas (Linhas 1, 2 e 3 da tabela) os escores obtidos são bons quando a análise é realizada em um ou em dois *datasets*, porém os resultados obtidos não são os melhores nos três cenários.

Tabela 5.2: Pontuação do DASRS Rest variando seus parâmetros utilizando o dataset NAB

Parâmetros				**Perfil**		
θ	**sequenceSize**	**restPeriod**	**Método Rest**	**Padrão**	**Pouco FP**	**Pouco FN**
7	3	probationaryPeriod / 5	B	67.184442	57.279468	72.375835
8	2	probationaryPeriod / 5	A	66.724499	58.616511	71.207137
8	2	probationaryPeriod / 5	B	66.558195	59.107571	70.808911
7	2	probationaryPeriod / 5	A	66.364190	60.224712	70.392219
7	3	probationaryPeriod / 5	A	66.295119	54.664718	72.070309
7	2	probationaryPeriod / 5	B	65.973788	60.259747	69.844595
9	2	probationaryPeriod / 5	B	65.642046	57.205879	70.198145
9	2	probationaryPeriod / 5	A	65.498529	56.088471	70.389824
10	2	probationaryPeriod / 5	A	65.309410	53.898435	70.838457
7	3	probationaryPeriod / 2	A	64.557534	56.508701	69.187782
7	2	2	B	64.204324	55.904979	68.952308
7	2	1	B	64.031550	55.559431	68.837125
8	2	2	B	63.891301	52.932797	69.318339
8	2	1	B	63.623699	52.397593	69.139937
8	3	probationaryPeriod / 5	B	63.448045	51.374637	69.310191
5	3	probationaryPeriod / 5	A	63.272767	55.400967	67.756557
8	2	probationaryPeriod / 2	A	62.919406	57.001613	66.658914
5	3	probationaryPeriod / 5	B	62.896974	55.485487	67.218673
7	2	probationaryPeriod / 2	A	62.763589	58.127593	66.267680
8	3	probationaryPeriod / 5	A	62.720126	47.197306	69.686981
10	2	probationaryPeriod / 5	B	62.403027	51.928756	67.464087
9	3	probationaryPeriod / 5	B	62.192615	49.975321	67.898525
7	4	probationaryPeriod / 5	A	62.169259	44.436809	69.894449
6	2	probationaryPeriod / 5	A	62.091904	56.779616	65.819890
6	3	probationaryPeriod / 5	A	61.735362	51.315248	67.018977
9	2	2	B	61.445084	47.922767	67.687527
6	2	probationaryPeriod / 5	B	61.406850	56.225348	65.075831
6	3	probationaryPeriod / 5	B	61.340937	52.032004	66.181314
9	2	1	B	61.224896	47.482391	67.540735
7	4	probationaryPeriod / 5	B	61.169606	48.646688	66.929163
10	3	probationaryPeriod / 5	A	61.069157	40.932873	69.448403
5	2	probationaryPeriod / 5	A	60.846639	56.495841	64.127644
6	2	2	B	60.825820	54.227183	64.975834
9	2	probationaryPeriod / 2	A	60.810040	53.786487	64.965314
5	3	2	B	60.742124	50.339683	66.069462
10	2	2	B	60.698050	44.336832	67.764217
6	2	1	B	60.683579	53.942700	64.881007
5	3	1	B	60.552469	49.960372	65.943026
7	2	0	-	60.351298	48.198928	66.383624
5	2	probationaryPeriod / 5	B	60.206174	56.030751	63.413312
6	4	probationaryPeriod / 5	A	60.164566	45.525199	66.833849
5	2	2	B	60.125472	55.053508	63.646866
7	5	probationaryPeriod / 5	A	60.124618	38.384193	69.106067
5	2	1	B	60.078058	54.958680	63.615257
5	4	probationaryPeriod / 5	B	60.057287	50.029455	65.038191
10	2	1	B	60.052146	43.045023	67.333614
9	3	probationaryPeriod / 5	A	59.926857	42.948812	67.250089
5	4	probationaryPeriod / 5	A	59.612604	47.498233	65.316449
8	2	0	-	59.275364	43.700924	66.241047
10	3	probationaryPeriod / 5	B	58.872504	44.514925	65.110405

A versão final do DASRS Rest utiliza método Rest A e os seguintes valores *default* dos parâmetros:

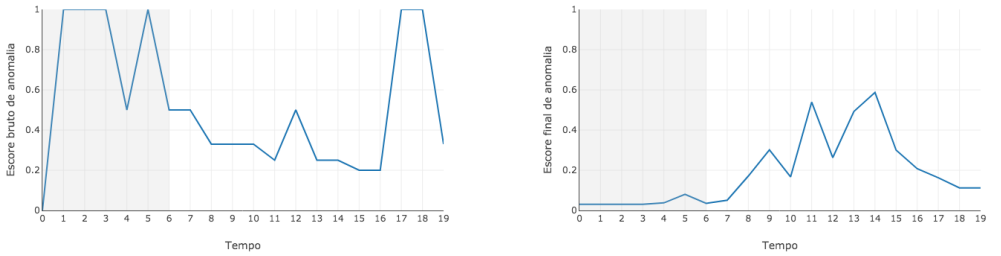
- θ: 7;
- $sequenceSize$: 2;
- $restPeriod$: $probationaryPeriod/5$.

5.4.2 DASRS Likelihood

Likelihood é uma medida de probabilidade do estado atual ser anômalo baseada no histórico dos escores brutos de anomalia calculados pelo algoritmo. A explicação detalhada do cálculo do escore *Likelihood* pode ser encontrada na seção 4.5.1.2.

Para criar a versão DASRS Likelihood, utilizamos a biblioteca NuPIC [Numenta, Inc., 2019.b], que foi desenvolvida pela mesma empresa que criou o NAB e o algoritmo Numenta HTM.

As figuras 5.6a e 5.6b ilustram respectivamente os escores bruto e final de anomalia gerados pelo DASRS Likelihood para a série temporal da figura 5.4a, com $\theta = 7$ e $sequenceSize = 2$. A área em cor cinza na figura representa o período de treinamento, quando o algoritmo está identificando os padrões existente na série.



(a) Escore bruto de anomalia (b) Escore final de anomalia - Likelihood

Figura 5.6: Escores de anomalia calculados pelo DASRS Likelihood

A tabela 5.3 mostra o passo a passo de execução do DASRS Likelihood para a série da figura 5.4a. Como acontece no DASRS Rest, a série temporal é representada pelas colunas Tempo e X. X continua representando o conjunto de valores escalares registrados ao longo do tempo, Ocorrências é quantidade de vezes que a Sequência foi encontrada, X' é o valor normalizado de X, Sequência são os últimos 2 ($sequenceSize$) elementos de X', o Escore Bruto é o escore bruto de anomalia calculado pelo algoritmo e o Escore Final é o escore $Likelihood$ calculado pela biblioteca NuPIC.

Tabela 5.3: Passo a passo de processamento do DASRS Likelihood

Tempo	X	X'	Sequência	Ocorrências	Escore Bruto	Escore Final - Likelihood
0	10,5	0	-	-	0	0,030
1	15,3	0	(0, 0)	1	1	0,030
2	23,2	1	(0, 1)	1	1	0,030
3	18,2	0	(1, 0)	1	1	0,030
4	27,8	1	(0, 1)	2	0,5	0,038
5	22,2	1	(1, 1)	1	1	0,080
6	20,0	0	(1, 0)	2	0,5	0,035
7	13,4	0	(0, 0)	2	0,5	0,051
8	19,0	0	(0, 0)	3	0,33	0,171
9	24,1	1	(0, 1)	3	0,33	0,301
10	20,9	0	(1, 0)	3	0,33	0,167
11	28,1	1	(0, 1)	4	0,25	0,539
12	22,9	1	(1, 1)	2	0,5	0,263
13	15,5	0	(1, 0)	4	0,25	0,492
14	10,4	0	(0, 0)	4	0,25	0,587
15	16,8	0	(0, 0)	5	0,2	0,300
16	24,0	1	(0, 1)	5	0,2	0,207
17	90,0	7	(1, 7)	1	1	0,162
18	28,9	1	(7, 1)	1	1	0,112
19	26,6	1	(1, 1)	3	0,33	0,112

O algoritmo 8 mostra os detalhes de implementação do DASRS. Na versão DASRS Rest o parâmetro $useLikelihood$ é definido como falso. Por outro lado, a versão DASRS Likelihood utiliza o parâmetro $useLikelihood$ como verdadeiro.

Incluímos um teste na versão Likelihood do algoritmo para identificar as anomalias pontuais, conforme mostrado na função $pointAnomaly$. Se uma anomalia pontual é identificada, o escore final retornado é igual a 1, independente dos valores calculados para o escore bruto e para o escore $Likelihood$. Isso foi realizado para compensar a limitação que esta técnica apresenta quando calcula a probabilidade do escore bruto ser realmente uma anomalia. Isso porque, apesar desta técnica diminuir a incidência de faltos positivos ela poderia também reduzir a quantidade de verdadeiros positivos encontrados pelo algoritmo.

Algoritmo 8: DASRS

Entrada:

- $minValue$: valor mínimo entre as observações
- $maxValue$: valor máximo entre as observações
- θ: quantidade de valores normalizados
- $sequenceSize$: tamanho da janela de sequencias
- $restPeriod$: período que o escore de anomalia é atenuado após ter detectado uma anomalia
- $useLikelihood$: valor boleando indicando se deve usar likelihood escore
- x_i: observação corrente

Saída: $AnomalyScore \in$ [0,1]: escore da anomalia

Function `init` $(minValue, maxValue, \theta, sequenceSize, restPeriod)$**:**
  $restWeakenFactor \leftarrow 0$
  $normInputSequence \leftarrow []$
  $sequences \leftarrow \{\}$
  $anomalyLikelihood \leftarrow nupic.algorithms.anomaly_likelihood()$
  $minVal \leftarrow null$
  $maxVal \leftarrow null$

Function `getRawAnomalyScore` $(occurrences)$**:**
  retorna $1/occurrences$

Function `getLikelihoodScore` $(currentScore)$**:**
  retorna $anomalyLikelihood.computeLogLikelihood(currentScore)$

Function `pointAnomaly` $(value)$**:**
  $isPointAnomaly \leftarrow False$
  se $minVal! = maxVal$ **então**
    $tolerance \leftarrow maxVal - minVal \times 0.05$
    $maxExpected \leftarrow maxVal + tolerance$
    $minExpected \leftarrow minVal - tolerance$
    se $value > maxExpectedorvalue < minExpected$ **então**
      $isPointAnomaly \leftarrow True$
  se $maxVal = nullorvalue > maxVal$ **então**
    $maxVal \leftarrow value$
  se $minVal = nullorvalue < minVal$ **então**
    $minVal \leftarrow value$
  retorna $isPointAnomaly$

Function `getRestScore` $(score)$**:**
  se $restWeakenFactor > 0$ **então**
    $score \leftarrow score/restWeakenFactor$
    $restWeakenFactor \leftarrow restWeakenFactor - 1$
  senão se $score \geq 1$ **então**
    $restWeakenFactor \leftarrow restPeriod$
  retorna $score$

Function `handleRecord` (x_i)**:**
  $normInpVal \leftarrow \lfloor \theta \times (x_i - minValue)/(maxValue - minValue) \rfloor$
  $normInputSequence.append(normInpVal)$
  se $len(normInputSequence) < sequenceSize$ **então**
    retorna 0
  se $normInputSequence \in sequences$ **então**
    $sequences[normInputSequence] += 1$
  senão
    $sequences[normInputSequence] = 1$
  $occurrences = count(normInputSequence\ in\ sequences)$
  $normInputSequence.pop(0)$
  $rawAnomalyScore \leftarrow getRawAnomalyScore(occurrences)$
  se $useLikelihood$ **então**
    $anomalyScore \leftarrow getLikelihoodScore(rawAnomalyScore)$
    se $pointAnomaly(x_i)$ **então**
      $anomalyScore \leftarrow 1$
  senão
    $anomalyScore \leftarrow getRestScore(rawAnomalyScore)$
  retorna $anomalyScore$

6
Experimentos e Resultados

Este capítulo descreve os experimentos realizados nesse trabalho e seus resultados. Utilizamos o *framework* NAB para executar os diversos algoritmos estudados. Os algoritmos buscaram anomalias no *dataset* nativo do NAB, no *dataset* Yahoo-A1Benchmark e em um terceiro *dataset* que criamos a partir de dados de monitoração reais coletados de centenas de servidores de produção hospedados no data center da Globo.com. Decidimos chamar esse *dataset* de Globo.com-dataset. A descrição detalhada dos *datasets* utilizados está na Seção 4.7. Além disso, executamos *benchmarks* de pontuação, métricas clássicas de desempenho, tempo de execução e utilização de memória para comparar os algoritmos desenvolvidos neste trabalho com diversos algoritmos de detecção de anomalia existentes na literatura.

Os resultados inseridos nas linhas em cor escura das tabelas pertencem aos algoritmos que criamos ou desenvolvemos. Os algoritmos DASRS são totalmente originais enquanto os outros são variações que desenvolvemos a partir dos algoritmos que já existiam.

6.1
Benchmark de Pontuação

Para o *benchmark* de pontuação, utilizamos os dados de pontuação calculado pelo NAB após execução dos algoritmos. Os detalhes da implementação do cálculo desta pontuação estão descritos na Seção 4.3. Os perfis de aplicação utilizados no cálculo da pontuação são os incluídos no NAB: perfil padrão, perfil recompensando pouco FP e perfil recompensando pouco FN. Os pesos de cada perfil estão descritos na tabela 4.1.

A pontuação obtida por cada algoritmo, em cada perfil, utilizando respectivamente os *datasets* NAB, Yahoo-A1Benchmark e o Globo.com-dataset está nas tabelas 6.1, 6.2 e 6.3. Ordenamos as tabelas em função da pontuação obtida no perfil padrão.

Tabela 6.1: Pontuação com o dataset NAB

	Pontuação por Perfil de Aplicação		
Detector	**Padrão**	**Recompensa Poucos FPs**	**Recompensa Poucos FNs**
Ensemble Detector	73.5	65.9	77.7
DASRS Likelihood	70.3	65.5	73.9
CAD OSE	69.9	67.0	73.2
Numenta	69.7	62.3	74.3
DASRS Rest	66.4	60.2	70.4
earthgecko Skyline	58.2	46.2	63.8
KNN CAD	58.0	43.4	64.8
Relative Entropy	54.1	48.0	57.9
Windowed Gaussian	40.1	23.9	47.7
Three-sigma rule	38.5	8.4	50.1
Etsy Skyline	35.7	27.1	44.5
Modified three-sigma rule	30.7	0.0	46.6
Tukey's test	26.7	0.0	41.9
Median Absolute Deviation	21.0	0.0	39.6
Bayesian Changepoint	17.7	5.1	32.3
EXPoSE	16.4	3.5	26.9

Tabela 6.2: Pontuação com o dataset Yahoo-A1Benchmark

	Pontuação por Perfil de Aplicação		
Detector	**Padrão**	**Recompensa Poucos FPs**	**Recompensa Poucos FNs**
DASRS Rest	67.1	62.6	72.3
CAD OSE	62.7	57.5	67.3
Ensemble Detector	55.3	49.8	60.7
KNN CAD	52.9	41.9	58.8
DASRS Likelihood	52.5	45.1	57.7
Windowed Gaussian	48.9	39.7	54.8
Relative Entropy	48.9	41.4	53.6
Three-sigma rule	47.5	39.9	52.1
Modified three-sigma rule	47.0	37.8	52.0
Numenta	46.2	41.5	50.3
Tukey's test	43.5	31.2	49.4
Bayesian Changepoint	41.1	23.9	53.7
Etsy Skyline	36.5	25.9	43.5
earthgecko Skyline	35.8	34.6	38.0
Median Absolute Deviation	27.1	0.0	38.2
EXPoSE	13.8	10.9	26.0

Tabela 6.3: Pontuação com o Globo.com-dataset

	Pontuação por Perfil de Aplicação		
Detector	**Padrão**	**Recompensa Poucos FPs**	**Recompensa Poucos FNs**
Numenta	63.7	52.7	72.4
Ensemble Detector	63.0	39.5	72.1
CAD OSE	62.7	49.1	72.0
DASRS Likelihood	60.6	50.9	66.6
DASRS Rest	55.0	23.2	67.8
KNN CAD	39.3	0.0	54.6
Etsy Skyline	21.0	0.0	31.9
Relative Entropy	17.3	0.0	33.2
EXPoSE	4.7	0.3	8.5
earthgecko Skyline	0.0	0.0	24.5
Windowed Gaussian	0.0	0.0	15.5
Three-sigma rule	0.0	0.0	5.1
Bayesian Changepoint	0.0	0.0	3.2
Modified three-sigma rule	0.0	0.0	1.6
Median Absolute Deviation	0.0	0.0	0.0
Tukey's test	0.0	0.0	0.0

Os resultados inseridos nas tabelas mostram que a algoritmo Ensemble Detector obtém o melhor resultado quando processamos o *dataset* NAB em busca de anomalias. No entanto, o Ensemble Detector não obteve a melhor pontuação quando analisou os *datasets* do Yahoo e o Globo.com-dataset, embora o algoritmo ainda esteja entre os primeiros. Isso acontece porque o modelo de classificação é criado a partir do *dataset* NAB, que é muito pequeno para criar um modelo genérico suficiente para atender diversas das características dos outros *datasets*.

Os algoritmos baseados em testes estatísticos não conseguiram ficar entre os de melhor pontuação em nenhum dos *datasets*, apesar de superarem a pontuação de alguns dos algoritmos pertencentes ao *benchmark* NAB original. A análise das tabelas também mostra que tais algoritmos registram seus melhores resultados ao processar o *dataset* Yahoo-A1Benchmark. O pior resultado ocorre quando o *dataset* processado é o Globo.com-dataset. Este resultado está relacionado com os tipos de anomalia predominantes em cada *dataset* e se as séries possuem comportamento predominantemente previsível ou predominantemente aleatório (imprevisível). O *dataset* Yahoo-A1Benchmark possui as séries com menor grau de comportamento aleatório. Suas anomalias são predominantemente do tipo pontual. Os outros *datasets* estudados possuem tanto anomalias pontuais quanto anomalias de contexto. As séries temporais do *dataset* Globo.com-dataset possuem grau de aleatoriedade maior que as do NAB e do Yahoo-A1Benchmark. Isso porque nós criamos o *dataset* utilizando métricas coletadas de servidores de produção de um data center real e a escolha da amostra de dados foi aleatória, enquanto os dois *datasets* seguiram alguns critérios para escolha da amostra de dados de forma a representar diversos tipos de anomalia. Além disso, o *dataset* NAB também incluí dados artificias. Portanto, é possível concluir que os algoritmos baseados em testes estatísticos obtêm bons resultados quando as anomalias a serem detectadas são do tipo pontual, embora não consigam obter a mesma qualidade de detecção quando as anomalias são do tipo anomalias de contexto.

O algoritmo DASRS Likelihood ficou em segundo lugar quando utilizamos o *dataset* NAB, perdendo apenas para o Ensemble Detector. O DASRS Rest ficou em quinto lugar, obtendo pontuação bem próxima às do CAD OSE e do Numenta (menos de 4 pontos de diferença), distante do sexto colocado (diferença de mais de 8 pontos pra o earthgecko Skyline). Quando o *dataset* Yahoo-A1Benchmark é processado, o DASRS Rest ficou na primeira colocação, com uma diferença de mais de 4 pontos para o segundo colocado. Neste *dataset* o DASRS Likelihood ficou em quinto lugar. Na análise do Globo.com-dataset o DASRS Likelihood e DASRS Rest alcançaram respectivamente o quarto e o quinto lugar. A versão Likelihood do DASRS mostrou ser a que lida melhor com séries temporais que possuem grande grau de aleatoriedade. Por isso esta versão obteve pontuação maior ao analisar tanto

o *datasets* NAB quanto o Globo.com-dataset.

6.2 Benchmark com as Métricas Clássicas de Desempenho

Nesta seção utilizamos algumas das métricas clássicas encontradas na literatura quando comparamos o desempenho dos algoritmos estudados:

– ***Precision*:** é a proporção de resultados positivos que são realmente positivos:

$$Precision = \frac{TP}{TP + FP} \tag{6-1}$$

– ***Recall*:** é a proporção de resultados positivos classificados como positivos:

$$Recall = \frac{TP}{TP + FN} \tag{6-2}$$

– ***F1-Score (F1)*:** é a média harmônica entre *precision* e *recall* e é dado pela equação abaixo:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6-3}$$

TP representa o total de anomalias detectadas corretamente. Apenas a primeira anomalia detectada dentro de uma janela foi considerada no cálculo do valor do TP, as demais anomalias detectadas na mesma janela foram ignoradas. FN representa o total de anomalias não detectadas. FP representa o total de anomalias detectadas incorretamente. As anomalias detectadas no período probatório foram ignoradas no cálculo do valor do FP.

As tabelas 6.4, 6.5 e 6.6 exibem os valores de TP, FP, FN, *Precision*, *Recall* e *F1-Score* para todos os *datasets* analisados. As tabelas estão ordenadas pela pontuação obtida na métrica *F1-Score*.

Tabela 6.4: Métricas clássicas utilizando o Dataset NAB

Detector	TP	FN	FP	Precision	Recall	F1-Score
CAD OSE	92	24	64	0.59	0.79	0.68
DASRS Likelihood	94	22	103	0.48	0.81	0.60
Ensemble Detector	101	15	139	0.42	0.87	0.57
DASRS Rest	91	25	135	0.40	0.78	0.53
Numenta	97	19	177	0.35	0.84	0.50
Relative Entropy	76	40	133	0.36	0.66	0.47
earthgecko Skyline	87	29	265	0.25	0.75	0.37
KNN CAD	91	25	314	0.22	0.78	0.35
Windowed Gaussian	73	43	399	0.15	0.63	0.25
Etsy Skyline	72	44	497	0.13	0.62	0.21
Three-sigma rule	85	31	646	0.12	0.73	0.20
Modified three-sigma rule	91	25	911	0.09	0.78	0.16
Tukey's test	84	32	882	0.09	0.72	0.16
EXPoSE	51	65	504	0.09	0.44	0.15
Bayesian Changepoint	67	49	766	0.08	0.58	0.14
Median Absolute Deviation	89	27	1111	0.07	0.77	0.14

Tabela 6.5: Métricas clássicas utilizando o Dataset Yahoo-A1Benchmark

Detector	TP	FN	FP	Precision	Recall	F1-Score
DASRS Rest	114	30	123	0.48	0.79	0.60
CAD OSE	110	34	160	0.41	0.76	0.53
earthgecko Skyline	61	83	33	0.65	0.42	0.51
Ensemble Detector	96	48	137	0.41	0.67	0.51
Numenta	83	61	159	0.34	0.58	0.43
DASRS Likelihood	96	48	210	0.31	0.67	0.43
Relative Entropy	91	53	212	0.30	0.63	0.41
Three-sigma rule	87	57	204	0.30	0.60	0.40
KNN CAD	99	45	290	0.25	0.69	0.37
Modified three-sigma rule	88	56	247	0.26	0.61	0.37
Windowed Gaussian	95	49	363	0.21	0.66	0.32
Tukey's test	87	57	328	0.21	0.60	0.31
Etsy Skyline	83	61	474	0.15	0.58	0.24
Bayesian Changepoint	109	35	737	0.13	0.76	0.22
Median Absolute Deviation	86	58	723	0.11	0.60	0.18
EXPoSE	35	109	223	0.14	0.24	0.17

Tabela 6.6: Métricas clássicas utilizando o Globo.com-dataset

Detector	TP	FN	FP	Precision	Recall	F1-Score
DASRS Likelihood	89	28	208	0.30	0.76	0.43
Numenta	105	12	438	0.19	0.90	0.32
CAD OSE	106	11	461	0.19	0.91	0.31
Ensemble Detector	111	6	545	0.17	0.95	0.29
DASRS Rest	109	8	684	0.14	0.93	0.24
KNN CAD	98	19	845	0.10	0.84	0.18
Etsy Skyline	63	54	631	0.09	0.54	0.16
Relative Entropy	76	41	926	0.08	0.65	0.14
EXPoSE	19	98	212	0.08	0.16	0.11
Bayesian Changepoint	0	117	0	-	-	-
Tukey's test	0	117	0	-	-	-
Three-sigma rule	0	117	0	-	-	-
Median Absolute Deviation	0	117	0	-	-	-
Modified three-sigma rule	0	117	0	-	-	-
earthgecko Skyline	0	117	0	-	-	-
Windowed Gaussian	0	117	0	-	-	-

O cálculo da pontuação NAB e as métricas clássicas de desempenho levam em consideração a quantidade de TPs, FPs e FNs. No caso da pontuação NAB, além dessas métricas, a localização dos TPs e FPs em relação as janelas de anomalia também é levada em consideração. De forma geral os resultados dos *benchmarks* de pontuação NAB foram semelhantes aos *benchmarks* com as métricas clássicas. Os algoritmos que obtiveram maior pontuação NAB também alcançaram os melhores resultados com as métricas clássicas.

A métrica *precision* não considera os FNs. Por isso, as anomalias não identificadas pelos algoritmos não reduziram o escore desta métrica. Por outro lado, a métrica *recall* não leva em consideração os FPs. Portanto, os algoritmos que identificam incorretamente muitas anomalias não são prejudicados no escore desta métrica. A métrica *F1-Score* obteve o resultado mais próximo ao da pontuação NAB por considerar os TPs, os FPs e os FNs.

A escolha de qual métrica deve ser utilizada para comparação dos algoritmos depende dos requisitos da aplicação. A melhor métrica de comparação será a *recall*, por exemplo, quando é essencial que um algoritmo encontre o maior número possível de anomalias verdadeiras (TP) sem ter problemas de identificação incorreta de anomalias (FP). Entretanto, a métrica *precision* é a melhor quando é um requisito da aplicação não detectar anomalias de forma incorreta (FP), mesmo que a detecção de anomalias verdadeiras (TP) seja prejudicada. Para as aplicações em que todas as medidas (TP, FP e FN) importam é melhor utilizar o *F1-Score* ou a pontuação NAB. A pontuação NAB traz a vantagem de levar em consideração a antecedência que a anomalia é detectada e permite ajustar os pesos dos perfis de aplicação de acordo com as necessidades específicas de cada aplicação.

6.3 Benchmark de Tempo de Execução

Para o *benchmark* de tempo de execução, utilizamos um computador com a seguinte configuração: MacBook Pro, Processador 3,1 GHz Intel Core i5, Memória 8 GB 2133 MHz LPDDR3, Sistema Operacional macOS High Sierra.

Realizamos dois experimentos para medir o tempo de execução dos algoritmos. No primeiro, rodamos os algoritmos utilizando o *framework* NAB e medimos o tempo para processar cada um dos *datasets*. No segundo, geramos uma série temporal com 10.000 observações e medimos o tempo de processamento a cada 1.000 observações analisadas.

6.3.1 Tempo de Execução Datasets NAB, Yahoo-A1Benchmark e Globo.com-dataset

No primeiro experimento para comparar o tempo de execução, utilizamos o *framework* NAB para executar os algoritmos estudados e os propostos nos *datasets* disponíveis. Medimos o tempo total gasto por cada algoritmo ao processar os *datasets*. Os resultados de 5 rodadas de testes estão nas tabelas 6.7, 6.8 e 6.9. O tempo de execução é medido em segundos.

Tabela 6.7: Tempo de execução utilizando o Dataset NAB

Detector	Rodada 1	Rodada 2	Rodada 3	Rodada 4	Rodada 5	Média
DASRS Rest	23	24	25	22	24	23,60
Three-sigma rule	33	35	34	34	34	34,00
Median Absolute Deviation	40	41	41	40	42	40,80
Tukey's test	45	43	47	44	43	44,40
Windowed Gaussian	45	49	49	45	47	47,00
Relative Entropy	62	65	66	63	63	63,80
DASRS Likelihood	64	66	68	63	63	64,80
Bayesian Changepoint	145	152	147	150	145	147,80
Modified three-sigma rule	189	204	190	207	197	197,40
EXPoSE	222	230	228	224	218	224,40
CAD OSE	289	295	303	287	282	291,20
Ensemble Detector	587	657	580	580	655	611,80
earthgecko Skyline	564	568	691	575	547	589,00
Numenta	696	768	738	791	688	736,20
KNN CAD	716	728	812	833	700	757,80
Etsy Skyline	16365	16072	16589	16497	16524	16409,40

Tabela 6.8: Tempo de execução utilizando o dataset Yahoo-A1Benchmark

Detector	Rodada 1	Rodada 2	Rodada 3	Rodada 4	Rodada 5	Média
DASRS Rest	7	8	8	8	9	8,00
Three-sigma rule	10	11	10	9	10	10,00
DASRS Likelihood	10	11	11	11	11	10,80
Tukey's test	12	12	12	11	13	12,00
Median Absolute Deviation	12	12	13	12	11	12,00
Windowed Gaussian	18	13	13	12	14	14,00
Relative Entropy	26	19	18	17	18	19,60
Bayesian Changepoint	25	22	20	20	21	21,60
Modified three-sigma rule	24	26	24	23	25	24,40
CAD OSE	56	53	52	54	53	53,60
earthgecko Skyline	74	61	61	61	61	63,60
KNN CAD	79	69	65	70	65	69,60
EXPoSE	81	66	70	62	63	68,40
Ensemble Detector	96	96	98	89	84	92,60
Numenta	212	188	181	192	178	190,20
Etsy Skyline	787	639	625	684	652	677,40

Tabela 6.9: Tempo de execução utilizando Globo.com-dataset

Detector	Rodada 1	Rodada 2	Rodada 3	Rodada 4	Rodada 5	Média
DASRS Rest	63	60	63	66	68	64,00
Three-sigma rule	82	82	86	88	83	84,20
Median Absolute Deviation	108	97	132	106	114	111,40
Tukey's test	111	113	117	105	121	113,40
Windowed Gaussian	139	129	141	130	164	140,60
DASRS Likelihood	153	131	153	133	136	141,20
Relative Entropy	196	184	197	187	189	190,60
Bayesian Changepoint	290	291	319	282	284	293,20
Modified three-sigma rule	532	675	577	508	533	565,00
EXPoSE	597	640	642	597	629	621,00
earthgecko Skyline	758	820	793	777	767	783,00
CAD OSE	927	1013	968	911	935	950,80
Ensemble Detector	1489	1582	1449	1543	1387	1490,00
KNN CAD	1787	1996	1976	1874	1955	1917,60
Numenta	1884	2020	2056	2093	2162	2043,00
Etsy Skyline	26654	27549	25985	26964	26749	26780,20

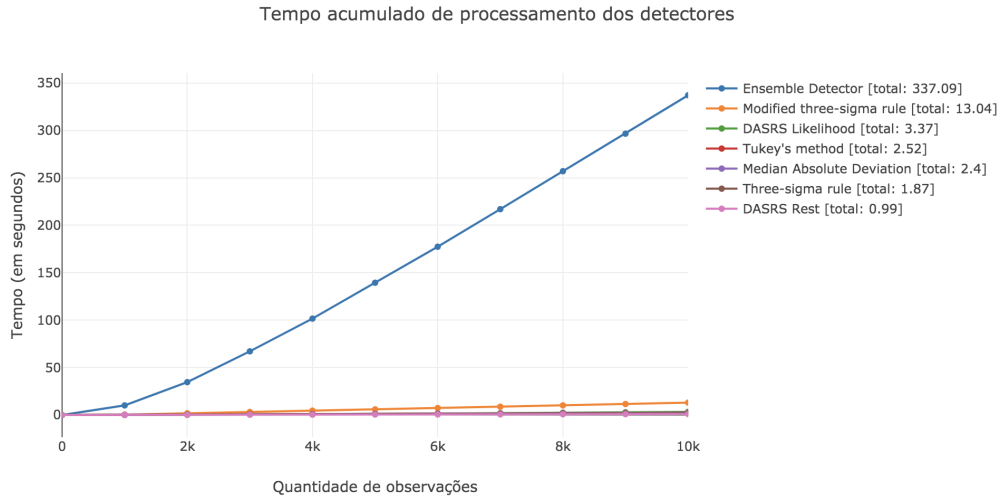
Os resultados mostram que o algoritmo DASRS Rest, proposto nesse trabalho, foi o mais rápido em todos os *datasets* testados. Os outros algoritmos desenvolvidos neste trabalho, exceto o Ensemble Detector, também apresentaram tempo pequeno de execução, superando a maior parte dos algoritmos da literatura presentes no *benchmark* NAB original.

6.3.2 Tempo de Execução Série Temporal com 10.000 Observações

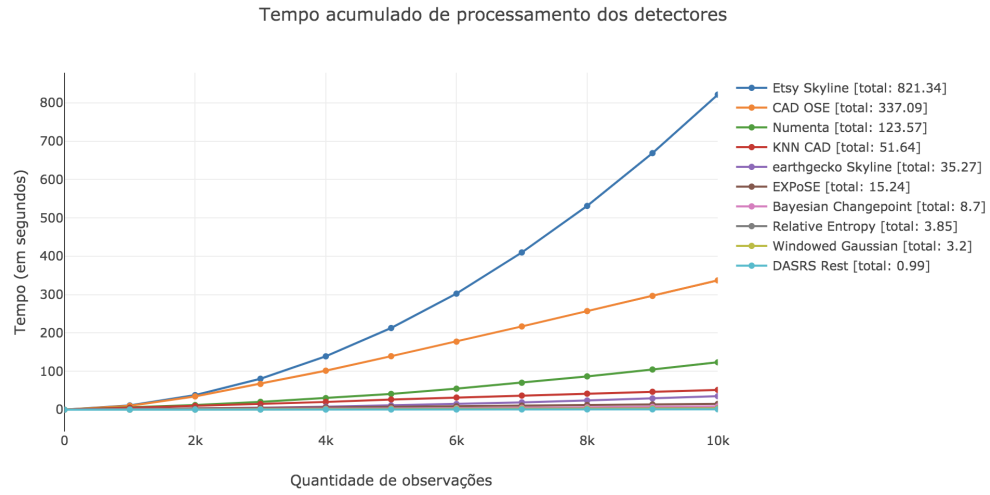
No segundo experimento para comparar o tempo de execução, utilizamos uma série temporal com 10.000 observações e medimos o tempo de processamento a cada 1.000 observações analisadas, para cada um dos algoritmos.

A figura 6.1a compara o tempo de processamento, em segundos, gasto pelos algoritmos desenvolvidos nesse trabalho. O DASRS Rest foi o que obteve melhor resultado, levando menos de 1 segundo para processar 10.000 observações. Three-sigma rule (1,87 segundos), Median Absolute Deviation (2,4 segundos), Tukey's method (2,52 segundos) e DASRS Likelihood (3,37 segundos) obtiveram bons resultados. Modified three-sigma rule demorou 13,04 segundos para processar as 10.000 observações e o Ensemble Detector teve o pior resultado, 337,09 segundos.

A figura 6.1b compara o tempo de processamento, em segundos, gasto pelos algoritmos incluídos no NAB e o DASRS Rest. Os resultados mostram que o DASRS Rest obteve o melhor resultado entre os algoritmos desenvolvidos nesse trabalho. Além disso, o DASRS Rest foi 3 vezes mais rápido que o melhor algoritmo do NAB. O resultado do Etsy Skyline foi muito ruim, 821,34 segundos para processar 10.000 observações. Os resultados do CAD OSE e do Numenta também foram ruins, 337,09 e 123,57, respectivamente.



(a) Detectores desenvolvidos neste trabalho



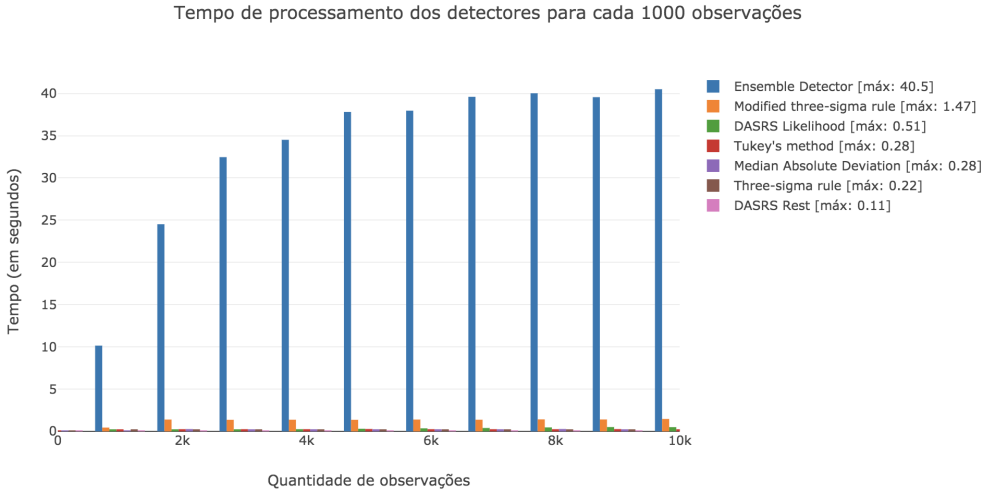
(b) DASRS Rest + Detectores incluídos no NAB

Figura 6.1: Tempo acumulado gasto pelos detectores no processamento de 10.000 observações

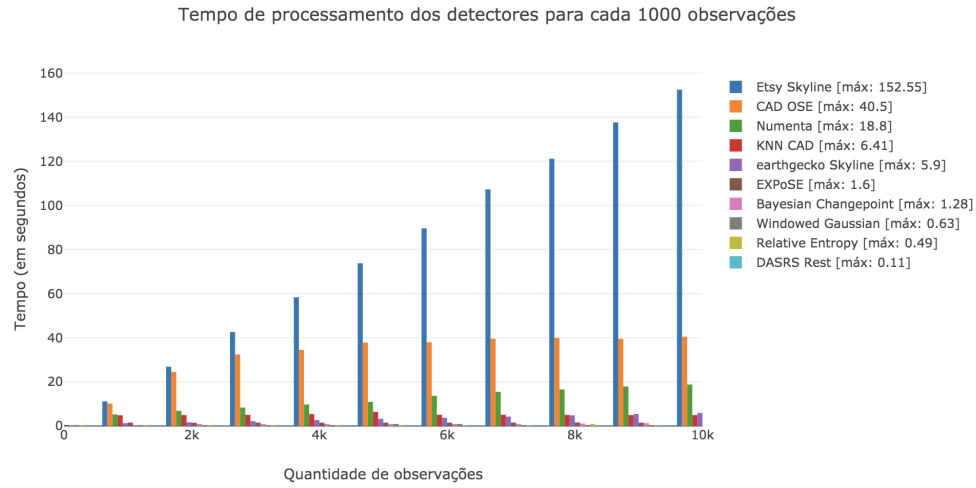
A análise dos resultados contidos nas figuras 6.1a e 6.1b mostram que alguns algoritmos ficam mais lentos à medida que mais dados são processados. A curvatura dos gráficos evidencia o problema nas figuras 6.2a e 6.2b. Também é possível visualizar esse problema quando analisamos a tabela 6.10 que exibe o tempo de processamento de cada intervalo com 1.000 observações. A tabela foi criada com os mesmos dados que geraram os gráficos das figuras citadas. É difícil observar o crescimento no tempo de processamento de todos os detectores através dos gráficos porque enquanto o mais eficiente gasta menos de 1 segundo o menos eficiente leva mais de 820 segundos para processar a mesma série de dados. Por causa da diferença de escala entre os resultados utilizamos a tabela para facilitar a visualização dos resultados.

A figura 6.2a compara o tempo entre os algoritmos desenvolvidos nesse trabalho. O Ensemble Detector teve o pior resultado, em que o tempo de processamento, de cada 1.000 observações, foi aumentando à medida que mais dados foram processados, até chegar na observação de número 8.000, onde se estabilizou. A partir daí o algoritmo passou a levar em torno de 40 segundos para analisar 1.000 observações. DASRS Likelihood também apresentou pequeno aumento no tempo de processamento à medida que mais dados foram processados, chegando a gastar 0,5 segundos para processar as últimas 1.000 observações. Modified three-sigma rule, Tukey's method, Median Absolute Deviation e Three-sigma rule alcançaram resultados semelhantes, com aumento no tempo de processamento das 1.000 primeiras observações e das 1.000 observações seguintes. Depois disso, eles quase não apresentaram variação no tempo de processamento. Modified three-sigma rule demora 1,47 segundos para processar 1.000 observações e os outros 3 algoritmos gastam entre 0,22 e 0,28 segundos. O algoritmo DASRS Rest, proposto nesse trabalho, obteve o melhor resultado. Ele apresentou o menor tempo de processamento, entre 0,09 e 0,11 segundos para processar cada 1.000 observações. Esta pequena variação não está relacionada com a quantidade de observações processadas e sim com o balanceamento de tempo dos processos feito pelo sistema operacional.

A figura 6.2b compara os algoritmos incluídos no NAB e o DASRS Rest. O DASRS Rest foi 4 vezes mais rápido que o melhor resultado obtido pelos algoritmos do NAB. Além disso, os algoritmos Etsy Skyline, Numenta, earthgecko Skyline apresentaram tempo de processamento crescente a cada iteração e não estabilizou até o processamento da observação de número 10.000. O CAD OSE também apresentou aumento no tempo de processamento, mas estabilizou após o processamento da observação 8.000.



(a) Detectores desenvolvidos neste trabalho



(b) DASRS + Detectores incluídos no NAB

Figura 6.2: Tempo gasto pelos detectores no processamento de 1.000 observações

Tabela 6.10: Tempo gasto pelos detectores no processamento de 1.000 observações

Detector	1K	2K	3K	4K	5K	6K	7K	8K	9K	10K	Total
DASRS Rest	0,11	0,09	0,10	0,10	0,09	0,10	0,09	0,10	0,10	0,10	0,99
Three-sigma rule	0,14	0,19	0,18	0,18	0,19	0,20	0,18	0,18	0,18	0,21	1,87
Median Absolute Deviation	0,12	0,25	0,28	0,24	0,25	0,25	0,25	0,28	0,24	0,25	2,40
Tukey's test	0,16	0,26	0,26	0,27	0,28	0,26	0,26	0,26	0,27	0,25	2,52
Windowed Gaussian	0,23	0,35	0,10	0,10	0,56	0,63	0,35	0,10	0,10	0,09	3,20
DASRS Likelihood	0,13	0,40	0,47	0,51	0,31	0,36	0,40	0,47	0,51	0,50	3,37
Relative Entropy	0,33	0,39	0,49	0,39	0,37	0,43	0,39	0,49	0,39	0,38	3,85
Bayesian Changepoint	0,41	0,98	1,13	1,28	0,80	0,90	0,98	1,13	1,28	1,21	8,70
Modified three-sigma rule	0,45	1,38	1,42	1,41	1,37	1,40	1,38	1,42	1,41	1,47	13,04
EXPoSE	1,55	1,54	1,60	1,52	1,52	1,50	1,54	1,60	1,52	1,52	15,24
earthgecko Skyline	1,30	4,26	4,83	5,43	3,23	3,72	4,26	4,83	5,43	5,90	35,27
KNN CAD	4,84	5,08	5,00	4,93	6,41	5,07	5,08	5,00	4,93	4,94	51,64
Numenta	5,22	15,46	16,58	17,92	10,93	13,68	15,46	16,58	17,92	18,80	123,57
CAD OSE	10,15	39,60	40,02	39,56	37,81	37,96	39,60	40,02	39,56	40,50	337,09
Ensemble Detector	10,15	39,60	40,02	39,56	37,81	37,96	39,60	40,02	39,56	40,50	337,09
Etsy Skyline	11,16	107,34	121,24	137,70	73,81	89,66	107,34	121,24	137,70	152,55	821,34

O DASRS Rest obteve o melhor resultado entre todos os algoritmos analisados. Além de apresentar o menor tempo de processamento, não houve aumento no tempo de processamento à medida que mais dados são analisados pelo algoritmo. O aumento no tempo de processamento pode inviabilizar a utilização de um algoritmo no processamento em *streaming*. Além disso, um algoritmo pode precisar processar simultaneamente milhares de séries temporais.

6.4 Benchmark de Memória Utilizada

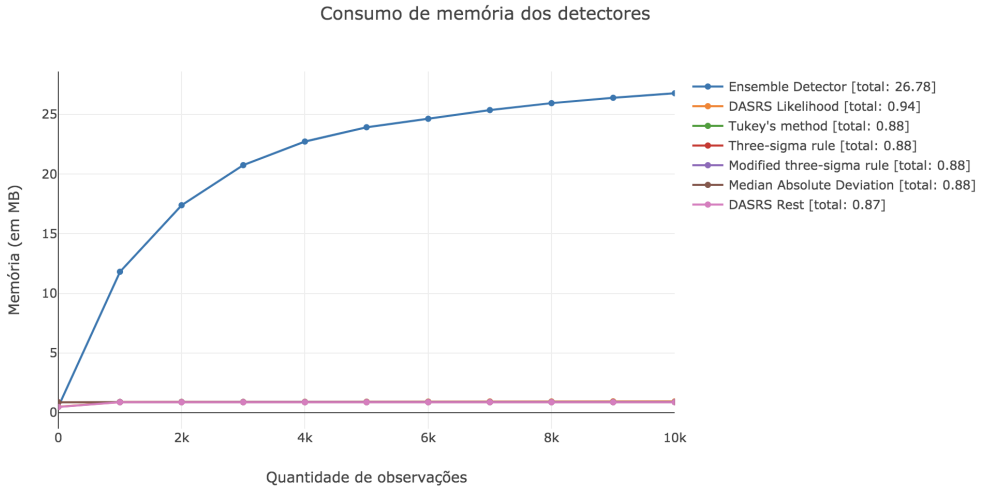
No *benchmark* de memória, usamos a função *asizeof* para avaliar a quantidade de memória utilizada pelos detectores de anomalias. Detalhes sobre a implementação desta função podem ser obtidos na página "ActiveState Code » Recipes"[1].

Utilizamos uma série temporal com 10.000 observações e medimos a memória utilizada por cada um dos detectores a cada 1.000 observações analisadas.

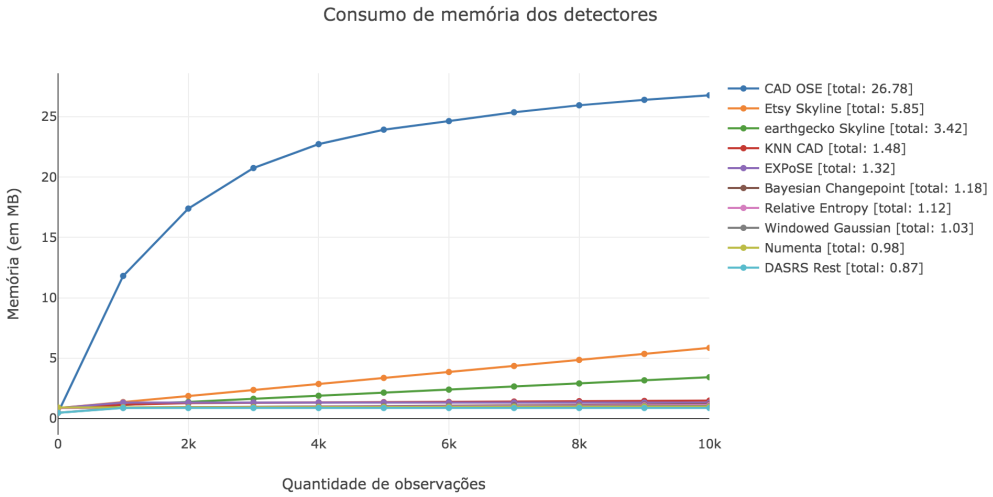
A figura 6.3a compara a memória entre os algoritmos desenvolvidos nesse trabalho. O Ensemble Detector teve pior resultado, consumindo mais de 30 vezes o que foi consumido pelos demais detectores, além de apresentar aumento da memória utilizada à medida que as observações são processadas. Os demais algoritmos, além de apresentarem consumo baixo da memória (menos de 1 MB), não apresentaram variação no consumo de memória.

A figura 6.3b compara a memória utilizada entre os algoritmos incluídos no NAB e o DASRS Rest, que foi o que teve melhor resultado entre os algoritmos desenvolvidos nesse trabalho. Os algoritmos CAD OSE, Etsy Skyline e earthgecko Skyline apresentaram aumento da memória utilizada e também tiveram consumo de memória maior que os demais.

[1] http://code.activestate.com/recipes/546530/



(a) Detectores desenvolvidos neste trabalho



(b) DASRS + Detectores incluídos no NAB

Figura 6.3: Memória utilizada pelos detectores no processamento de 10.000 observações

Os resultados mostram que o DASRS Rest não apresentou variação no consumo de memória. Além disso, o DASRS Rest é o algoritmo com menor consumo de memória (0,87 MB) entre todos os algoritmos analisados. O pouco uso da memória é um fator muito importante na escolha de um detector de anomalias. Principalmente em um data center, onde é necessário analisar métricas de milhares de máquinas ao mesmo tempo. Também é importante não ter aumento de memória utilizada à medida que os dados são processados pelo detector, pois os mesmos serão usados para processamento *streaming*.

7
Aplicação na Globo.com utilizando o DASRS

Este capítulo descreve como o DASRS (versão Rest) foi utilizado em uma aplicação real, chamada Sofia Anomaly Detector, criada para analisar métricas das máquinas virtuais que são administradas pelos administradores de banco de dados da Globo.com[1].

7.1
Descrição do Ambiente

O Sofia Anomaly Detector foi configurado para analisar métricas de um subconjunto de máquinas administradas pelo time de banco de dados da Globo.com, que são as VMs criadas pelo DBaaS[2]. O DBaaS (*Database as a Service*) é uma aplicação para gerenciamento e automação no provisionamento de bancos de dados. Atualmente o DBaaS provisiona bancos de dados dos seguintes tipos: Redis[3], MySQL[4] e MongoDB[5]. Os bancos de dados podem atender aplicações de desenvolvimento ou de produção. Em produção todos os bancos possuem alta disponibilidade. Assim, cada *cluster* de banco de dados é configurado com no mínimo duas VMs e os dados são replicados da instância *master* (ou primário no MongoDB) para a instância *slave* (ou secundário no MongoDB).

Atualmente o DBaaS administra aproximadamente 2.800 VMs. A quantidade de máquinas não é exata porque o ambiente é dinâmico e o número de VMs pode variar à medida que bancos de dados são criados ou removidos. Além de dinâmico, o ambiente é diversificado. O tamanho das VMs varia entre VMs com 1 CPU e 500 MB de memória até VMs com 16 CPUs e 32 GB de memória. A utilização do banco e consequente consumo dos recursos das máquinas também é muito variado. Algumas VMs ficam ociosas na maior parte do tempo e outras operam no limite dos recursos disponíveis. Algumas máquinas apresentam variação de consumo dependendo da hora do dia ou dia da semana e outras apresentam variação do consumo ao longo do tempo devido a diversos fatores, como por exemplo, aumento de utilização do banco de dados devido ao crescimento ou sucesso da aplicação.

[1] https://www.globo.com
[2] https://github.com/globocom/database-as-a-service
[3] https://redis.io/
[4] https://www.mysql.com/
[5] https://www.mongodb.com/

Entre as atividades dos DBAs do time de banco de dados, umas das mais importantes é monitorar os bancos para mantê-los disponíveis o máximo de tempo possível. Todas as métricas das VMs e dos bancos de dados são coletadas periodicamente e alarmes são disparados quando os *thresholds* previamente configurados são atingidos. Para não gerar muitos faltos alarmes, os *thresholds* geralmente são configurados com valores altos. Por esse motivo, alguns alarmes são disparados apenas quando o banco de dados já se encontra indisponível. O objetivo de utilizar detecção de anomalias nas métricas das VMs é criar alarmes mais inteligentes capazes de notificar uma anomalia assertivamente antes que o serviço fique indisponível.

7.2 Arquitetura do Sistema

O Sofia Anomaly Detector é o sistema responsável por organizar o fluxo de dados no processo de detecção de anomalias. Ele coleta as métricas, orquestra a análise das mesmas pelo detector e oferece painéis de visualização dos resultados. Os *dashboards* de alarmes do Sofia oferecem gráficos das métricas e das anomalias detectadas.

Construímos o Sofia utilizando diversos componentes. A coleta das métricas é feita pelo Telegraf[6]. O Kafka[7] é utilizado para armazenamento temporário das métricas e é utilizado pelas aplicações que precisam processar estas métricas em tempo real. Os dados são persistidos no banco TimeScale[8]. Os gráficos são exibidos através de *dashboards* no Grafana[9]. Os alarmes podem ser visualizados através de painéis no DBMonitor. O Status do detector (objeto serializado) é armazenado num banco Redis, desta forma pode-se reiniciar o sistema sem que o detector precise passar novamente pelo período probatório (período de aprendizagem).

A figura 7.1 apresenta a arquitetura do sistema, que utiliza o DASRS Rest para detectar anomalias das métricas coletadas. A descrição detalhada de todos os componentes é feita em seguida.

[6]https://www.influxdata.com/time-series-platform/telegraf/
[7]https://kafka.apache.org/
[8]https://www.timescale.com/
[9]https://grafana.com/

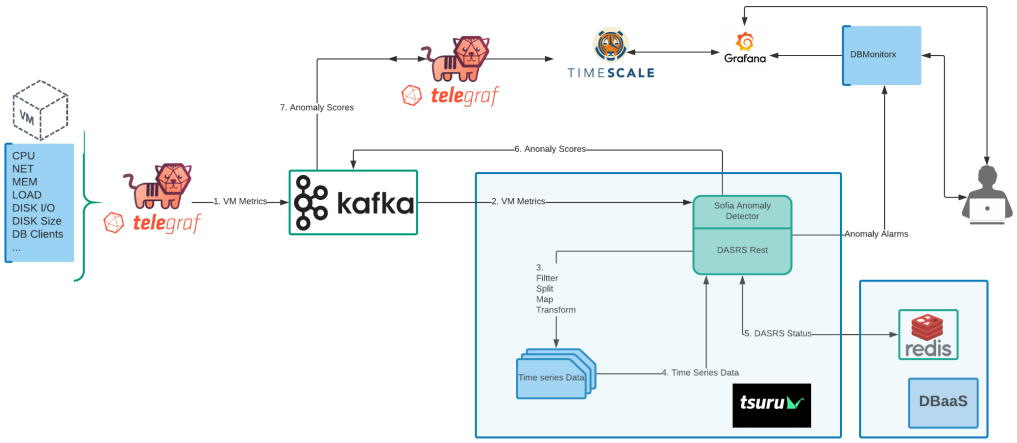


Figura 7.1: Arquitetura do Sofia Anomaly Detector

Descrição dos componentes utilizados:

– **Telegraf** é um agente para coleta, agregação, processamento e envio de métricas. Através de *plugins*, possui integração com uma grande variedade de sistemas, tanto para coleta quanto para envio das métricas. Na arquitetura apresentada, o Telegraf foi instalado nas VMs para coletar métricas e enviar para o Kafka. O Telegraf também foi utilizado para consumir mensagens do Kafka e enviar para o banco de dados TimeScale.

– **Kafka** é uma plataforma distribuída de mensagens e *streaming*. O Kafka permite publicar, armazenar, processar e consumir mensagens. Suporta um volume grande de fluxo de dados. Todas as métricas coletadas das VMs são publicadas no Kafka, onde ficam disponíveis para serem consumidas por diversas aplicações.

– **Tsuru**[10] é um PaaS (*Platform as a Service*) extensível e de código aberto. O Tsuru permite que desenvolvedores desenvolvam suas aplicações sem precisar se preocupar com servidores ou em como executar o *deploy*. O Tsuru oferece um cliente que permite gerenciar a aplicação desenvolvida por linhas de comandos. É possível, por exemplo, adicionar bancos de dados à aplicação desenvolvida. O Sofia Anomaly Detector foi criado utilizando o PaaS Tsuru.

– **Redis** é um banco de dados *NoSQL* do tipo chave-valor. O Redis é extremamente rápido, tanto para escrita como para leitura dos dados. As instâncias do DASRS são serializadas e salvas no Redis, para que não seja necessário reprocessamento caso o sistema seja reiniciado.

– **DBaaS** é uma aplicação para provisionamento de banco de dados. Possui integração com o Tsuru e permite que o desenvolvedor crie um banco de

[10]https://tsuru.io/

dados para sua aplicação através de API. O DBaaS foi utilizado para criar o banco de dados Redis utilizado pela aplicação.

– **TimeScale** é um banco de dados relacional projetado para armazenar dados de séries temporais, fornecendo particionamento automático no tempo. Foi projetado a partir do PostgreSQL[11] e possui suporte completo ao SQL. Os escores de anomalia gerados pelo DASRS são armazenados no TimeScale.

– **Grafana** é uma plataforma para visualizar e analisar métricas por meio de gráficos. Possui suporte para diversos tipos de bancos de dados, incluindo o PostgreSQL. O Grafana foi utilizado nesse sistema para criar gráficos das métricas coletadas juntamente com as anomalias detectadas.

– **DBMonitor** é uma ferramenta de monitoração e administração de banco de dados desenvolvida pelo time de banco de dados da Globo.com. O DBMonitor é responsável por monitorar todos os bancos administrados pelo time. É uma ferramenta interna, não está disponível no mercado. O DBMonitor foi utilizado nesse sistema para exibir painéis com os alarmes das anomalias detectadas.

7.3 Detalhes da Implementação

O Telegraf foi instalado nas máquinas e configurado para coletar métricas do sistema operacional como utilização de CPU, memória, *swap*, *load*, tráfego de rede. Métricas de bancos de dados como quantidade de sessões e quantidade de operações realizadas (*queries, updates, deletes, inserts*) também são coletadas pelo Telegraf. Configuramos o intervalo de coleta para 1 minuto. As métricas coletadas são enviadas para o Kafka.

A aplicação Sofia Anomaly Detector consome as mensagens publicadas no Kafka e executa algumas operações para transformar a mensagem consumida no formato esperado pelo detector de anomalia. Estas operações correspondem aos *steps* 3 e 4 da figura 7.1. A primeira operação realizada é um filtro, pois o Telegraf coleta um conjunto grande de métricas e na atual versão do sistema estamos interessados em analisar apenas 5 métricas: *cpu user*, *cpu system*, *cpu idle*, *cpu iowait* e *used percent disk*. As métricas de disco são de todos os discos da VM. Por isso, um novo filtro é aplicado, pois estamos interessados apenas no disco onde estão os dados do banco de dados. As métricas de CPU vêm numa única mensagem, então essa mensagem é dividida em 4 séries de dados correspondentes as métricas de CPU citadas acima. Nesta etapa também é criado um identificador único para as séries de dados criadas, contento o nome da máquina e nome da métrica.

[11] https://www.postgresql.org/

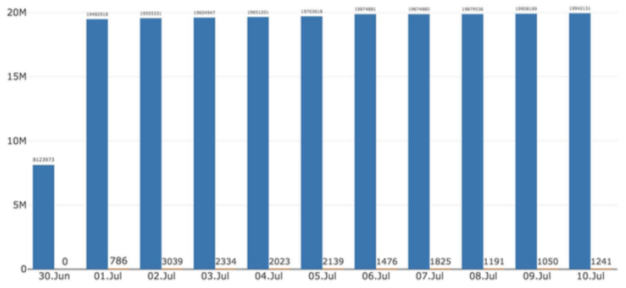
Após o *step* 4, o dado analisado faz parte de uma série temporal e contém as seguintes informações: identificador da série de dados, valor escalar da métrica e *timestamp* que a coleta foi realizada. O identificador é utilizado para buscar no banco Redis o objeto serializado. Caso não encontre significa que é a primeira observação desta série de dados então uma nova instância do detector DASRS é criado. Caso o objeto tenha sido encontrado significa que outras observações desta série de dados já foram analisadas, então a instância do DASRS é carregada a partir do seu dado serializado. O DASRS analisa a observação corrente e gera um escore de anomalia. Este resultado é publicado numa outra fila do Kafka que posteriormente será lida pelo Telegraf e persistida no banco TimeScale. Dependendo do valor do escore de anomalia um alarme é disparado no DBMonitor para que o DBA seja notificado.

O DBA também tem acesso ao Grafana, que foi utilizado para criar *dashboards* com os gráficos das métricas coletadas juntamente com as anomalias detectadas.

7.4 Algumas Estatísticas do Sistema

- O sistema está analisando 14.000 métricas por minuto (2.800 VMS, 5 métricas de cada VM);
- O tempo aproximado para processamento de cada observação é de apenas 0,001348 segundos (*steps* 2 até o 6 da figura 7.1);
- O tempo entre a coleta da métrica e visualização do resultado do detector no Grafana é de menos 1 minuto. O sistema não está apresentando atrasos e o tempo entre geração da métrica e visualização do resultado do detector no Grafana é devido ao intervalo de *flush* do Telegraf;
- O banco de dados Redis está utilizando apenas 12 MB de memória para guardar 14.000 instâncias do DASRS.
- Aproximadamente 20.000.000 de observações são analisadas por dia e destas, aproximadamente 0,005% são reconhecidas como anomalia pelo sistema. Mais detalhes podem ser vistos na figura 7.2).

O Sofia Anomaly Detector foi implantado no data center de produção da Globo.com no dia 30 de junho de 2019. A figura 7.2 mostra a quantidade de observações analisadas e a quantidade de anomalias detectadas entre 30 de junho e 10 de julho. As primeiras 1.440 observações de cada série (correspondente ao período de 24 horas) foram consideradas como período de treinamento e as anomalias detectadas neste período foram ignoradas.



Data	Observações	Anomalias
30.Jun	8.123.973	0
01.Jul	19.482.918	786
02.Jul	19.555.331	3.039
03.Jul	19.604.947	2.334
04.Jul	19.651.201	2.023
05.Jul	19.703.618	2.139
06.Jul	19.874.881	1.476
07.Jul	19.874.880	1.825
08.Jul	19.879.536	1.191
09.jul	19.908.169	1.050
10.jul	19.942.131	1.241
Total	**205.601.585**	**17.104**

Figura 7.2: Observações analisadas X Anomalias detectadas

8
Conclusão e Trabalhos Futuros

Esta dissertação propõe um algoritmo eficiente de detecção de anomalia chamado DASRS. O algoritmo proposto contabiliza as sequências normalizadas dos dados e gera um escore de anomalia inversamente proporcional a quantidade de vezes que a sequência é identificada. O DASRS possui duas versões propostas. Chamamos a primeira de DASRS Rest e a segunda de DASRS Likelihood. A dissertação também discute outros cinco algoritmos de detecção de anomalias para séries temporais. Quatro deles, Three-sigma rule, Median Absolute Deviation, Modified three-sigma e Tukey's method, são baseados em testes estatísticos clássicos para identificação de anomalias. Também apresentamos o Ensemble Detector, um algoritmo que combina os resultados de um conjunto de detectores para fornecer resultados mais precisos que os detectores deste conjunto.

Os algoritmos baseados em testes estatísticos foram bastantes eficientes no tempo de processamento e apresentaram pouco consumo de memória. Não foi observado aumento do consumo de memória nem no tempo de execução, mesmo nos testes com uma quantidade maior de observações. Apesar da eficiência no tempo de processamento, estes algoritmos não obtiveram bons resultados na pontuação calculada pelo *framework* NAB nem nas métricas clássicas de desempenho. Mostramos que os algoritmos baseados em testes estatísticos são bons para detectar anomalias pontuais, mas não são eficientes para detectar anomalias de contexto.

Os primeiros colocados no *benchmark* original do NAB são os algoritmos CAD OSE, Numenta, earthgecko Skyline e KNN CAD. Todos eles apresentam tempo de processamento muito alto. Com exceção do KNN CAD, estes algoritmos também apresentaram aumento no tempo de processamento à medida que a quantidade de observações analisadas cresce. Já em relação com a memória utilizada, todos, exceto o Numenta, apresentam aumento no consumo de memória à medida que a quantidade de observações analisadas aumenta.

O Ensemble Detector obteve ótimos resultados em relação a pontuação NAB e métricas clássicas de desempenho. Ele ficou em primeiro lugar quando os testes foram feitos no *dataset* do NAB e entre os três primeiros nos outros dois *datasets*. Ter ficado em primeiro lugar no *dataset* do NAB não foi nenhuma surpresa, pois este *dataset* foi utilizado na fase de treinamento do algoritmo. Não ter repetido a

primeira colocação nos outros *datasets* significa que o algoritmo não conseguiu criar um modelo genérico o suficiente. Isso aconteceu porque os *datasets* utilizados na fase de treinamento não possuem todos os comportamentos possíveis. O tempo de execução e a quantidade de memória utilizada pelo Ensemble Detector dependem da classificação da série temporal e correspondem a um dos detectores, do *benchmark* do NAB, utilizados no treinamento.

Os resultados da avaliação de desempenho mostram que, apesar dos bons resultados de pontuação apresentados, não é recomendável utilizar o Ensemble Detector quando a análise de métricas precisa ser realizada em tempo real. Este detector é mais complexo que os demais, pois precisa inicialmente identificar a classe que a série temporal pertence para então poder instanciar o detector real que irá analisar os dados da série. Além disso, pode ocorrer degradação de desempenho, em função do tipo de detector escolhido pelo algoritmo.

As duas versões do DASRS obtiveram ótimos resultados na pontuação, no tempo de processamento e no consumo de memória. Entre os algoritmos avaliados, o DASRS Rest apresentou o menor consumo de memória e o menor tempo de processamento. Além disso, ele não apresenta degradação, mesmo quando processa séries temporais que possuem grande quantidade de observações. O DASRS Likelihood obteve consumo de memória pequeno, apesar de ser um pouco mais lento que o DASRS Rest e apresentar um pequeno aumento no tempo de processamento quando analisa séries temporais com grande quantidade de observações. O DASRS Likelihood ficou em segundo lugar na pontuação utilizando o *dataset* do NAB, perdendo apenas para o Ensemble Detector. O DASRS Rest ficou na primeira colocação utilizando o *dataset* Yahoo-A1Benchmark. O DASRS Likelihood e o DASRS Rest obtiveram a quarta e a quinta colocação, respectivamente, quando processaram o *dataset* Globo.com-dataset. Além disso, os dois estão entre os cinco primeiros colocados em todos os *datasets* testados.

Um data center possui um ambiente dinâmico e diversificado, com um volume muito grande de dados para serem analisados. O algoritmo para detectar anomalias nestes dados precisa ser eficiente no consumo de memória e tempo de processamento e não requerer configuração ou ajustes manuais de parâmetros. As duas versões do DASRS atendem estes requisitos, além de terem obtido ótimos resultados de pontuação NAB e nas métricas clássicas de medição de desempenho.

O DASRS não está restrito a métricas de monitoração de máquinas de um data center. Ele pode ser utilizado para análise de qualquer sequência de dados numéricos, por exemplo, métricas geradas por sensores ou por dispositivos de IoT (*Internet of Things*). O algoritmo não requer treinamento nem configuração manual de parâmetros. Por isso, pode ser facilmente implementado para processar múltiplos tipos de series temporais geradas por em diferentes tipos de dispositivos.

Implementamos o DASRS Rest no ambiente de produção da Globo.com. Para isso, criamos uma aplicação chamada Sofia Anomaly Detector. A aplicação foi criada para organizar o fluxo dos dados que são enviados para o DASRS, que então analisa os valores das observações. O detector analisou 14.000 métricas por minuto de 2.800 máquinas. O Sofia Anomaly Detector consegue coletar, processar os dados e exibir os resultados da análise de cada métrica em menos de 1 minuto.

8.1 Trabalhos Futuros

Como trabalhos futuros planejamos melhorias na aplicação Sofia Anomaly Detector visando aumentar sua capacidade de processamento e diminuir a quantidade de alarmes a partir das anomalias detectadas.

A versão atual do Sofia Anomaly Detector faz análise das observações de forma sequencial. A cada minuto a aplicação lê 14.000 observações. Em seguida, executa uma instância do DASRS para analisar sequencialmente cada observação coletada. A aplicação não apresentou atrasos quando processou essa quantidade de observações. No entanto, acreditamos que podem ocorrer atrasos se a quantidade de observações para processar for muito grande. Como melhoria, pretendemos rodar a análise da observação pela aplicação de forma paralela, aumentando a capacidade do sistema para processar uma quantidade muito maior de observações.

O Sofia Anomaly Detector implementado no ambiente de produção da Globo.com reportou aproximadamente 1.000 anomalias, que corresponde a 0,005% das observações analisadas. Apesar do percentual estar dentro do que é esperado, segundo as regras empírica e *three-sigma* (ver seção 5.2.1), acreditamos que o número total de anomalias detectadas ainda é grande. Por isso, ainda pode inviabilizar a geração do alarme, que ainda precisaria ser analisado por um analista (DBA).

Observando as anomalias detectadas pelo sistema, verificamos que muitas delas não são causadas por problemas, como os listados na seção 1.1 e que precisam de uma intervenção do administrador. As figuras 8.1a e 8.1b mostram dois exemplos de anomalias identificadas pelo DASRS e reportadas pelo Sofia Anomaly Detector durante o processamento das métricas *CPU User* e *CPU Idle*. Duas máquinas foram analisadas. Podemos observar pelo gráfico que os pontos em vermelho nas figuras são anomalias e que o gráfico voltou ao status normal logo em seguida. Observamos que este comportamento é comum em muitas máquinas analisadas. Elas podem ficar ociosas, sem apresentar consumo de CPU, por longos períodos, e em determinado momento apresentar um pico de consumo e voltar a ficar ociosa logo em seguida. O pico de consumo neste cenário é realmente uma anomalia, mas como o consumo voltou ao normal logo em seguida, não houve indisponibilidade ou impacto nas aplicações que rodam na máquina.

(a) Anomalia detectada na métrica CPU User (b) Anomalia detectada na métrica CPU Idle

Figura 8.1: Exemplo de anomalias detectadas no Sofia Anomaly Detector

Como melhoria, em lugar de gerar um alarme para cada anomalia detectada, pretendemos criar um processo que irá analisar as anomalias reportadas pelo sistema juntamente com os dados da série de temporal. Primeiro para um intervalo de tempo anterior e depois para um intervalo de tempo posterior à anomalia reportada. Assim, o alarme será gerado apenas se houver mudança significativa nas características da série temporal nos dois intervalos analisados.

Referências bibliográficas

[Adams & MacKay 2007] ADAMS, R. P.; MACKAY, D. J.. **Bayesian online changepoint detection**. Technical report, University of Cambridge, 2007. 3, 4.6.7

[Ahmad & Hawkins, 2016] AHMAD, S.; HAWKINS, J.. **How do neurons operate on sparse distributed representations? a mathematical theory of sparsity, neurons and active dendrites**. 01 2016. 4.5.1

[Ahmad & Lavin, 2015a] AHMAD, S.; LAVIN, A.. **Evaluating real-time anomaly detection algorithms - the numenta anomaly benchmark**. CoRR, abs/1510.03336, 2015. (document), 1.3, 2.4, 3, 4, 4.2, 4.3, 4.3

[Ahmad & Lavin, 2015b] AHMAD, S.; LAVIN, A.. **The numenta anomaly benchmark [white paper]**. 4.7.1

[Ahmad & Purdy, 2016] AHMAD, S.; PURDY, S.. **Real-time anomaly detection for streaming analytics**. CoRR, abs/1607.02480, 2016. 3, 4.5.1, 4.5.1.3

[Ahmad et al., 2017] AHMAD, S.; LAVIN, A.; PURDY, S. ; AGHA, Z.. **Unsupervised real-time anomaly detection for streaming data**. Neurocomputing, 06 2017. (document), 3, 4.4, 4.5.1, 4.5, 4.6, 4.5.1.3, 5.4

[Brockwell & Davis, 2016] BROCKWELL, P. J.; DAVIS, R. A.. **Introduction to Time Series and Forecasting**. Springer Texts in Statistics. Springer International Publishing, 2016. 2.1

[Burnaev & Ishimtsev, 2016] BURNAEV, E.; ISHIMTSEV, V.. **Conformalized density- and distance-based anomaly detection in time-series data**. arXiv:1608.04585, 2016. 3, 4.6.1

[Chandola et al., 2009] CHANDOLA, V.; BANERJEE, A. ; KUMAR, V.. **Anomaly detection: A survey**. ACM Comput. Surv., 41(3):15:1–15:58, 07 2009. 2.4, 2.4, 2.4, 2.5, 2.6, 3

[Earthgecko, 2019] EARTHGECKO. **Earthgecko skyline [online code repository]**. https://github.com/earthgecko/skyline. Accessed: 2019-03-01. 3, 4.5.3

[Etsy, Inc., 2013] ETSY, I.. **Skyline [online code repository]**. https://github.com/etsy/skyline. Accessed: 2019-03-01. 3, 4.6.6

[Fox, 1972] FOX, A. J.. **Outliers in time series**. Journal of the Royal Statistical Society. Series B, 34:350–363, 1972. 3

[Gabel, 2013] GABEL, M.. **Unsupervised anomaly detection in large datacenters**. Master's thesis, Israel Institute of Technology, 2013. 1.1

[Guha et al. 2016] GUHA, S.; MISHRA, N.; ROY, G. ; SCHRIJVERS, O.. **Robust random cut forest based anomaly detection on streams**. In: PROCEEDINGS OF THE 33RD INTERNATIONAL CONFERENCE ON INTERNATIONAL CONFERENCE ON MACHINE LEARNING - VOLUME 48, ICML'16, p. 2712–2721. JMLR.org, 2016. 3, 4.6.3

[Hawkins & Ahmad, 2016] HAWKINS, J.; AHMAD, S.. **Why neurons have thousands of synapses, a theory of sequence memory in neocortex**. Frontiers in Neural Circuits, 10:23, 2016. 4.5.1

[Hawkins & Blakeslee, 2004] HAWKINS, J.; BLAKESLEE, S.. **On Intelligence**. Times Books, New York, NY, USA, 2004. 4.5.1

[Hekimoglu & Koch, 2000] HEKIMOGLU, S.; KOCH, K. R.. **How can reliability of the test for outliers be measured?** Allgemeine Vermessungs-Nachrichten, 107. Jg.:247–253, 2000. 5.2.1

[Iglewicz & Hoaglin, 1993] IGLEWICZ, B.; HOAGLIN, D. C.. **How to detect and handle outliers**. American Society for Quality Control, Milwaukee, 1993. 5.2.3, 5.2.3

[Jeffrey, 2008] WOOLDRIDGE, J.. **Introductory Econometrics: A Modern Approach (with Economic Applications, Data Sets, Student Solutions Manual Printed Access Card)**. South-Western College Pub, 2008. 2.1

[Karagiannidis & Lioumpas, 2007] KARAGIANNIDIS, G. K.; LIOUMPAS, A. S.. **An improved approximation for the gaussian q-function**. IEEE Communications Letters, 11(8):644–646, 08 2007. 4.5.1.2

[Kejariwal, 2015] KEJARIWAL, A.. **Twitter engineering: Introducing practical and robust anomaly detection in a time series [online**

blog]. https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html. Accessed: 2019-03-01. 3, 4.6.4

[Laxhammar & Göran 2015] LAXHAMMAR, R.; FALKMAN, G.. **Inductive conformal anomaly detection for sequential detection of anomalous sub-trajectories**. Annals of Mathematics and Artificial Intelligence, 74(1-2):67–94, 06 2015. 4.6.1

[Leys et al. 2013] LEYS, C.; LEY, C.; KLEIN, O.; BERNARD, P. ; LICATA, L.. **Detecting outliers: do not use standard deviation around the mean, use absolute deviation around the median**. JOURNAL OF EXPERIMENTAL SOCIAL PSYCHOLOGY, 49(4):764–766, 2013. 5.2.2

[Miller, 1991] MILLER, J.. **Short report: Reaction time analysis with outlier exclusion: Bias varies with sample size**. The Quarterly Journal of Experimental Psychology Section A, 43(4):907–912, 1991. 5.2.2

[Numenta, Inc., 2016] NUMENTA, I.. **The numenta anomaly benchmark competition for real-time anomaly detection**. https://numenta.com/blog/2016/08/10/numenta-anomaly-benchmark-nab-competition-2016-winners/. Accessed: 2019-03-01. 4.5.2, 4.6.1

[Numenta, Inc., 2019.a] NUMENTA, I.. **NAB: Numenta anomaly benchmark [online code repository]**. https://github.com/numenta/NAB. Accessed: 2019-03-01. 4.2

[Numenta, Inc., 2019.b] NUMENTA, I.. **Nupic: Numenta platform for intelligent computing [online code repository]**. https://github.com/numenta/nupic. Accessed: 2019-03-01. 5.4.2

[Rajaraman & Ullman, 2011] RAJARAMAN, A.; ULLMAN, J. D.. **Mining of Massive Datasets**. Cambridge University Press, New York, NY, USA, 2011. 2.2

[Rosner, 1983] ROSNER, B.. **Percentage points for a generalized esd many-outlier procedure**. Technometrics, 25:165–172, 1983. 3, 4.6.4

[Ross, 2004] ROSS, S. M.. **Introduction to Probability and Statistics for Engineers and Scientists**. Elsevier Academic, London, UK, third edition, 2004. 5.2.1

[Schneider et al., 2016] SCHNEIDER, M.; ERTEL, W. ; RAMOS, F. T.. **Expected similarity estimation for large-scale batch and streaming anomaly detection**. CoRR, abs/1601.06602, 2016. 2.4, 3, 4.6.8

[Smirnov, 2016] SMIRNOV, M.. **Contextual anomaly detector [online code repository]**. https://github.com/smirmik/CAD, 2016. Accessed: 2019-03-01. 3, 4.5.2

[Tukey, 1977] TUKEY, J. W.. **Exploratory Data Analysis**. Addison-Wesley, 1977. 5.2.4

[Twitter, Inc., 2015] TWITTER, I.. **Twitter anomalydetection r package [online code repository]**. https://github.com/twitter/AnomalyDetection. Accessed: 2019-03-01. 4.6.4

[Wang et al. 2011] WANG, C.; VISWANATHAN, K.; CHOUDUR, L.; TALWAR, V.; SATTERFIELD, W. ; SCHWAN, K.. **Statistical techniques for online anomaly detection in data centers**. In: 12TH IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT (IM 2011) AND WORKSHOPS, p. 385–392, 05 2011. 3, 4.6.2

[Wilson, 2019] WILSON, G.. **Earthgecko skyline [online documentation]**. https://earthgecko-skyline.readthedocs.io/en/latest/overview.html. Accessed: 2019-03-01. 4.5.3

[Yahoo! Webscope, 2019] **Yahoo! webscope dataset ydata-labeled-time-series-anomalies-v1_0**. https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70. Accessed: 2019-03-01. 1.3, 4.7.2

[Žliobaitė et al., 2014] GAMA, J.; ŽLIOBAITĖ, I.; BIFET, A.; PECHENIZKIY, M. ; BOUCHACHIA, A.. **A survey on concept drift adaptation**. ACM Comput. Surv., 46(4):44:1–44:37, 03 2014. 2.3