UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

CIÊNCIA DA COMPUTAÇÃO

Gabriel Zancheta Scavazini

Detecção De Abuso Em Dns: Verificação De Concept Drift E Automação De Retreino Utilizando Apache Airflow

São José Do Rio Preto

2023

Gabriel Zancheta Scavazini

Detecção De Abuso Em Dns: Verificação De Concept Drift E Automação De Retreino Utilizando Apache Airflow

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título Bacharel em Ciência da Computação, junto ao Departamento de Ciências de Computação e Estatística, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiador: NIC.br - Proc. 2764/2018

Orientador: Adriano Mauro Cansian

São José Do Rio Preto

2023

S288d

Scavazini, Gabriel Zancheta
Detecção De Abuso Em Dns: Verificação De Concept Drift E Automação De Retreino Utilizando Apache Airflow / Gabriel Zancheta Scavazini. -- São José do Rio Preto, 2023
34 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Adriano Mauro Cansian

1. Redes de computadores. 2. Nomes de domínio na internet. 3. Aprendizado de máquina. 4. Desvio de conceito. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Gabriel Zancheta Scavazini

Detecção De Abuso Em Dns: Verificação De Concept Drift E Automação De Retreino Utilizando Apache Airflow

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título Bacharel em Ciência da Computação, junto ao Departamento de Ciências de Computação e Estatística, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Banca Examinadora

Prof. Dr. Adriano Mauro Cansian
UNESP – São José do Rio Preto
Orientador

Profª. Drª. Rogéria Cristiane Gratão de Souza
UNESP – São José do Rio Preto

Prof. Dr. Leandro Neves
UNESP – São José do Rio Preto

São José Do Rio Preto

14 de Novembro de 2023

Este trabalho é dedicado aos meus cachorros Marley Zancheta Scavazini e Joaquim José Zancheta Scavazini 🐾.

Agradecimentos

Aos meus pais, Celso Scavazini e Meire Luci Zancheta Scavazini que prestaram suporte durante toda minha vida, proporcionando um ensino de qualidade e uma vida saudável.

A minha irmã, Sabrina Zancheta Scavazini, a qual, incondicionalmente, prestou suporte a todo momento durante minha graduação.

Ao meu orientador, professor e amigo Adriano Mauro Cansian, pela oportunidade de iniciação científica e aconselhamento durante momentos difíceis.

Ao time ACME!, em especial Guilherme, Vitor, Marcos e Vinícios, pela amizade e tempo cedido para ajuda na criação de um tema de monografia.

Para todos os meus professores, amigos e colegas do curso de Ciência da Computação, pelo grande conhecimento repassado e ajuda em todos os momentos da graduação.

Este trabalho foi financiado por intermédio do Convênio de Pesquisa e Inovação com o NIC.BR - Núcleo de Informações e Coordenação do Ponto BR, Processo Fundunesp número 2764/2018 - NIC. BR.

Resumo

O sistema de DNS é uma estrutura crucial na internet mundial, e são organizados de forma hierárquica a partir de sistemas distribuidos. No Brasil, que conta com mais de 5 milhões de domínios registrados, abusos como DNS *Hijacking*, Cache *Poisoning* e amplificação de DNS podem ocorrer diariamente. Para combater tais ações, práticas de coleta de dados passivos são utilizadas, que, por sua vez, são processadas por meio de *machine learning* para detectar e parar abusos automaticamente. Contudo, o modelo de aprendizado de máquina utilizado atualmente não é incremental e com o passar do tempo gera *concept drift*. Então, para identificar o *concept drift*, são utilizados métodos estatísticos ou baseados em janelas. Quando detectados, o controle de fluxo por meio de Apache Airflow, um gerenciador de workflow, aciona o retreinamento de máquina por meio de DAGs (grafo acíclico orientado). Portanto, criando assim um fluxo automatizado de execução tornando o modelo incremental.

Abstract

The DNS system is a crucial structure on the worldwide internet, organized hierarchically through distributed systems. In Brazil, which has over 5 million registered domains, abuses such as DNS hijacking, cache poisoning, and DNS amplification can occur daily. To combat such actions, passive data collection practices are employed, which are then processed through machine learning to automatically detect and stop abuses. However, the current machine learning model is not incremental and over time generates concept drift. Therefore, to identify concept drift, statistical or window-based methods are used. When detected, flow control through Apache Airflow, a workflow manager, triggers machine retraining through DAGs(directed acyclic graph). Thus, creating an automated execution flow, making the model incremental.

Lista de Figuras

Listings

Sumário

1 Introdução

Uma das infraestruturas mais importantes da internet mundial é o *Domain Name Server* (DNS) pois realiza a resolução de nomes de domínios para seus respectivos endereços IP descrito em RFC (1987). Em setembro de 2022 o registrar NIC.br (2022) atingiu a marca de 5 milhões de domínios registrados, além de oferecer 1,5 milhão de registros com *DNSSEC*. Contudo, é importante destacar que uma tecnologia como essa pode ser facilmente utilizada de forma abusiva, como em casos de DNS *Hijacking* (sequestro de DNS), distribuição de *malwares* e *phishing*. Surge, então, a necessidade de detectar e neutralizar tais ameaças.

Atualmente, o modelo utilizado para a detecção de abusos no DNS opera em um padrão estático. Ou seja, o algoritmo foi treinado apenas uma vez e não possui a capacidade de se adaptar dinamicamente às mudanças do ambiente.

Em Bayram, Ahmed e Kassler (2022), diversas metodologias para testes de *concept drift* são exploradas, e grande parte desse estudo será incorporada no desenvolvimento desta monografia.

No entanto, o artigo Togbe et al. (2021) apresenta melhorias significativas ao adotar uma abordagem evolutiva. No estudo, os conceitos de aprendizado de máquina foram revisitados e testados para detectar *concept drift*, utilizando o modelo KSwin. Especificamente, o KSwin foi ajustado para ser compatível com o modelo necessário para essa detecção.

Nesse contexto, as mudanças propostas no KSwin são essenciais para acompanhar as transformações dinâmicas no ambiente digital. No modelo atualmente em uso, que é supervisionado, essas alterações podem não ser necessárias, dadas as particularidades do método pelo qual o modelo foi construído.

Nessa monografia serão exploradas a utilização de técnicas de *machine learning* já implementadas anteriormente em Silva (2022), juntamente com a automação de retreino, utilizando conceitos de *concept drift* e ferramentas como *Apache Airflow*.

Objetivo

Esta monografia propõe a evolução dos projetos de Gardini (2022) e Silva (2022) aprimorando a utilização de detecção de abuso em DNS por meio de *machine learning*.

Projetos anteriores adotaram uma abordagem de aprendizado de máquina estática, deixando de lado a necessária adaptação ao longo do tempo. Além disso, a evolução constante de temas, como abusos em DNS, faz com que o modelo rapidamente se desvie de sua eficácia inicial.

Portanto, os principais aspectos a serem abordados são:

- Verificar e detectar periodicamente a presença de *concept drift* dentro dos *dataframes* treinados para detecção de abuso em DNS
- Criar e utilizar *DAGs* a partir da ferramenta *Apache Airflow* para garatir um *worflow* automatizado de retreino.
- Comprovar a eficácia de um modelo incremental dentro do contexto de Abuso em DNS.

Para que os objetivos sejam alcançados, há a necessidade metodológica de calcular e explicitar o *concept drift* utilizando a linguagem computacional *Python* para que seja de fácil relação com a ferramenta *Apache Airflow* e sua criação de *DAGs* que utilizam da mesma linguagem.

2 Fundamentação Teórica

Alguns conceitos são necessários para a aplicabilidade do projeto, dentre eles o entendimento de DNS, *concept drift*, *Apache Airflow* e o funcionamento de uma API de treinamento, como a utilizada em Gardini (2022)

2.1 Sistema de nomes de domínio

O DNS é o sistema que permite a resolução de nomes de domínio em seu endereco IP correspondente. Logo, ao invés dos usuários decorarem cada um dos IP o qual acessam os sites, podem utilizar *'example.com'*

Segundo Kurose e Ross (2017) o funcionamento básico do DNS envolve três principais componentes:

a. Servidores DNS

   Armazenam e fornecem informações sobre os nomes de domínio e seus respectivos endereços IP. Os servidores DNS possuem um formato hierárquico de operação como demonstrado na Figura 1, seu servidor raiz *(root)* é o primeiro a ser acessado, seguido por TLD (*Top Level Domain*), ccTLD (*country code Top Level Domain*) e seus devidos servidores autoritativos. Cada servidor contém arquivos de zona a qual mapeiam em partes os nomes de domínio.

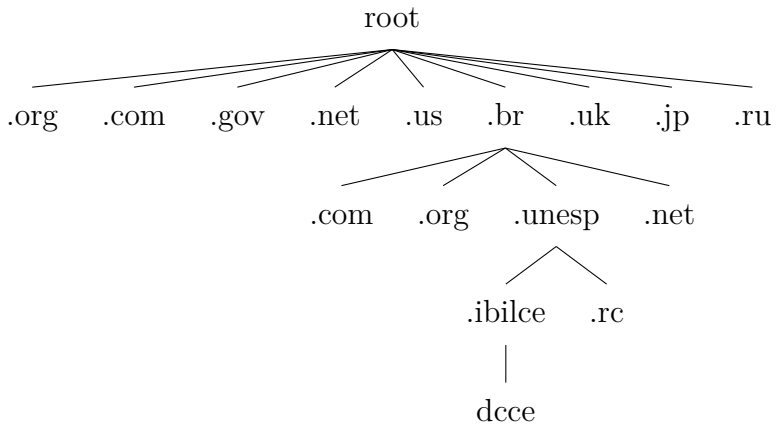
b. *Resolvers*

Eles fazem parte do sistema do cliente e são responsáveis por enviar requisições DNS para os servidores DNS. Os resolvers podem estar embutidos em sistemas operacionais ou serem fornecidos pelos provedores de internet (ISPs). Eles são responsáveis por encaminhar as consultas e receber as respostas. Além disso, podem armazenar as consultas em cache para agilizar consultas futuras.

c. Registros DNS

Também chamados de Registros de Zona, são divisões lógicas de domínios que se organizam de forma hierárquica. Cada zona é administrada por um servidor DNS autoritativo, que armazena informações sobre os nomes de domínio e seus respectivos endereços IP relacionados. Esses registros existem para facilitar e organizar as informações do DNS, permitindo que o sistema distribuído seja escalável.

Figura 1: Hierarquia DNS



Fonte: Autor.

Além disso a execução da tradução ocorre em uma série de eventos que podem ser descritas em ordem:

1. Solicitação de resolução: Ao digitar *'www.example.com'* o computador envia uma solicitação de resolução de DNS para o servidor autoritativo DNS.

2. Caches locais: O computador então verifica se o domínio e seu endereço IP correspondente estão armazenados em seu cache. Tal ação ajuda a diminuir o tempo de resposta para o usuário, pois as respostas de DNS recentes podem estar armazenadas em forma temporária.

3. Consulta ao servidor DNS: Caso o endereço desejado não esteja em cache local, o computador enviará uma consulta DNS para o servidor configurado, seja automaticamente pelo próprio sistema operacional, manualmente pelo usuário ou o servidor de DNS padrão de seu ISP.

4. Consulta recursiva: Com a consulta recebida ao servidor DNS ocorre a tentativa de resolução de forma hierárquica demonstrada na Figura 1, ou seja, o servidor DNS A consulta outros servidores B e C na tentativa de resolver o nome de domínio, o processo se repete até que seja encontrado ou que seja determinado que o domínio não existe.

5. Resposta DNS: Com a resposta correta e endereço IP correspondente ao domínio da consulta, é então retornada a informação ao computador solicitante.

6. Cache de servidor DNS: O servidor que realizou o processo de resolução armazena o resultado para agilizar consultas futuras.

7. Conexão estabelecida: Já com o endereço IP obtido pela consulta DNS, o computador pode então realizar a conexão com o servidor web.

2.1.1 DNS Passivo

De acordo com a fonte CISCO (2022), o DNS passivo (pDNS) é a captura contínua de informações de um servidor DNS como demonstrado na Figura 2. Sua principal função é permitir a análise de dados que possivelmente foram modificados, possibilitando a detecção e a classificação de anomalias potencialmente maliciosas.

O DNS passivo desempenha um importante papel na detecção de atividades suspeitas. Ao contrário das abordagens ativas, em que ocorrem consultas diretas de DNS, o DNS passivo permite a análise sem alertar os atores maliciosos sobre a investigação em andamento. Tornando essa abordagem de extrema importância para se obter informações sobre o abuso e contribuir diretamente para a segurança da rede.

Cabe mencionar que os dados capturados pelo pDNS devem passar por um processo de anonimização, pois, são considerados de cunho sensível e podem identificar o usuário que realizou a consulta DNS.

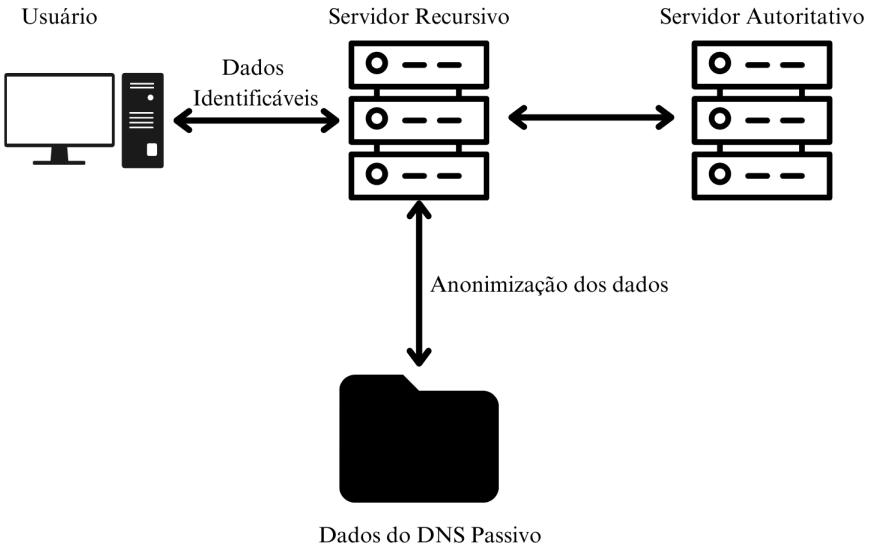
2.1.2 Abuso em DNS

O abuso em DNS é uma categoria que visa amplamente afetar a estrutura do DNS fazendo com que funcione de maneira indesejada. Tais atividades podem ocasionar instabilidade e ter consequências significativas para a segurança do DNS.

A definição formal de abuso em DNS, conforme mencionado no site ICANN (2023), é a seguinte:

"Qualquer atividade maliciosa com o objetivo de interromper a infraestrutura do DNS ou fazer com que o DNS opere de maneira não intencional. Atividades abusivas incluem corromper os dados da zona DNS, obter controle administrativo de um servidor de nomes

Figura 2: Exemplo de pDNS



Fonte: Autor.

e inundar o DNS com milhares de mensagens para degradar os serviços de resolução de nomes."

Os exemplos comuns de abuso em DNS incluem:

Cache Poisoning

O envenenamento de cache ocorre quando um atacante insere informações falsas em um servidor DNS. Por exemplo, o atacante H pode substituir o valor original D pelo valor C, fazendo com que o servidor as envie para os usuários que solicitarem dados desse domínio, que por sua vez recebem tais informações falsas.

DNS Hijacking

O sequestro de DNS envolve a interceptação e o redirecionamento ilegal do tráfego DNS. Por exemplo, o atacante H pode redirecionar o usuário A de B para C, fazendo com que o usuário acesse um destino diferente do pretendido. Comprometendo assim a parte essencial do DNS, fazendo com que a vítima possa contrair *malwares* ou caia em *phishing*.

DNS Amplification

A amplificação de DNS ocorre quando um servidor DNS mal configurado gera tráfego excessivo para um alvo específico. O atacante H pode manipular o servidor D para enviar uma grande quantidade de tráfego ao destino B, fazendo com que a resolução de nomes fique comprometida.

Com o passar do tempo técnicas que são consideradas atuais, como as apresentadas, podem já não ser utilizadas. Cabe, então, a criação de um modelo de aprendizado que

evolua com novas metodologias e abusos.

2.2 Concept Drift

O artigo Bayram, Ahmed e Kassler (2022) define *concept drift* como uma definição probabilística que descreve um conceito que varia ao longo do tempo. Sendo definida como a distribuição conjunta de X (um vetor de características) e y (a resposta).

Ademais, o *concept drift* está intrinsecamente relacionado com um aprendizado em fluxo de dados, onde ocorre uma sequência contínua de elementos em um período de tempo indeterminado, de forma sequencial. Assim, o *concept drift* está para o aprendizado de fluxo assim como a mudança de conjunto está para o aprendizado em lote.

Formalmente, é definido como a mudança da distribuição conjunta em dois momentos, t e $t+w$, sendo t um ponto de tempo (ou um período) e w a janela de tempo que está sendo verificada. Sendo assim, há *concept drift* quando a distribuição $P_t(X, y)$ e $P_{t+w}(X, y)$ não são iguais.

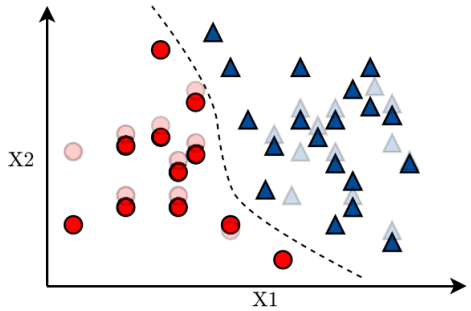
Segundo o modelo Bayesiano de decisão, a distribuição conjunta pode ser descrita na Equação 1.

$$P_t(X, y) = P_t(y|X) \times P_t(X) = P_t(X|y) \times P_t(y) \tag{1}$$

Onde $P_t(X|y)$ denota a distribuição de probabilidade posterior, $P_t(X)$ é a probabilidade dos dados de entrada, $P_t(y)$ é a distribuição de probabilidade a priori das etiquetas de destino e $P_t(X|y)$ é a distribuição de densidade de probabilidade condicional de classe.

O deslocamento de conceito pode ocorrer de diversas formas, como mostrado na Figura 3

Figura 3: Tipos de Drift



Fonte: Bayram, Ahmed e Kassler (2022).

As equações que geram o gráfico Figura 3 e são definidas pela Equação 2,Equação 3 e Equação 4. Outros tipos mais severos de deslocamento podem ser representados pela Equação 5 e vistos no Figura 4.

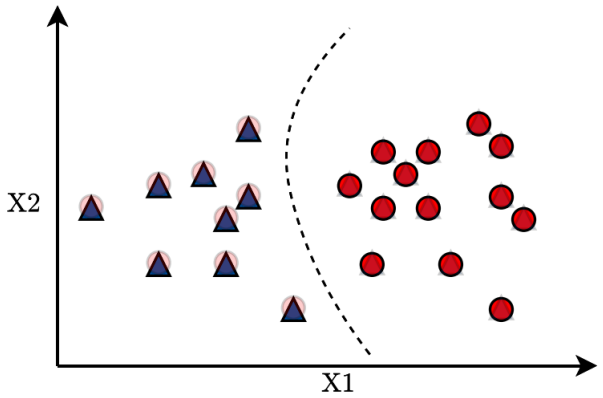
$$P_t(y|X) \neq P_{t+w}(y|X) \tag{2}$$

$$P_t(X) \neq P_{t+w}(X) \tag{3}$$

$$P_t(y) \neq P_{t+w}(y) \tag{4}$$

$$\forall x(argmax P_t(y|X) = c_1 \ \ \& \ \ argmax P_{t+w}(y|X) = c_2) \tag{5}$$

Figura 4: Exemplo de Drift Severo



Fonte: Bayram, Ahmed e Kassler (2022).

Detecção de *Concept Drift*

Vários métodos eficientes existem para a detecção de *concept drift* e precisam ser determinados conforme a necessidade e especialidade do código aplicado. Podemos separar os métodos para deteção de *concept drift* em duas categorias: baseados em estatística e baseados em janelas (GONÇALVES et al., 2014).

Métodos baseados em estatísticas

Analisam estatísticas e características dos dados para detectar mudanças no conceito. Alguns exemplos são:

- Análise de densidade de kernel: Compara as densidades de probabilidade das amostras de dados em intervalos de tempo para detectar mudanças de comportamento
- Teste de Friedman-Rafsky: Compara previsões de modelos treinados com diferentes conjuntos de dados. Então compara e verifica o desempenho dos modelos, verificando se há mudança de conceito.

Métodos baseados em janelas

Monitoram o desempenho do modelo em janela de dados, utilizam métricas estatísticas para detectar mudanças.

- ADWIN(Adaptive Windowing): Ajusta automaticamente a janela de acordo com as mudanças ocorridas. Utiliza testes estatísticos para detectar mudanças significativas no resultado.
- DDM (Drift Detection Method): Utiliza o teste do Qui-Quadrado para comparar o desempenho atual com o inicial, sendo assim, quando a taxa de erro se torna relativamente alta indica a incidência de *drift*.
- KSWIN: Utiliza janela deslizante para detecção de *drift*, leva em consideração o contexto local para detectar mudanças, eficaz para mudanças sazonais ou variações temporais.

2.3 Apache Airflow

Plataforma *open-source* (Apache (2023)) para criação e monitoramento de *workflows* orientados a lote. Permite conectar-se a praticamente qualquer outra tecnologia devido ao seu funcionamento em Python. Sua principal característica é que todos os fluxos também são controlados por códigos em Python, chamados de DAGs (*directed acyclic graph*)

2.3.1 Vantagens do Apache Airflow

Sua *framework* contém operadores preparados para se conectarem com diversas tecnologias. Além disso, a execução do código para verificação de *concept drift* pode ser agendada em períodos de tempos, permitindo que o Apache Airflow mantenha o controle automatizado de execução e monitoramento de falhas.

Dentre suas vantagens podemos citar:

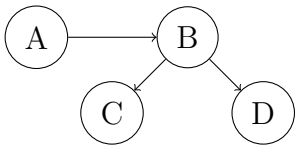
- Código em Python, facilitando a escrita e manutenção de fluxos de trabalho.
- Integração com sistema de versionamentos, como o Git, permitindo de forma eficiente o controle de mudanças e histórico de alterações nos seus fluxos.
- Utilização de sistemas já em funcionamento, possibilitando a integração com infraestruturas já existentes e a reutilização de componentes.

2.3.2 DAGs (Directed Acyclic Graph)

DAGs, ou Gráficos Acíclicos Direcionados, são coleções de todas as tarefas que devem ser executadas, em ordem. Os resultados de cada tarefa impactam na execução de outras

tarefas. Por exemplo na Figura 5, a tarefa A pode ser seguida apenas pela tarefa B. No entanto, dependendo do resultado da tarefa B, o fluxo pode seguir para as tarefas C ou D.

Figura 5: Grafo Acíclico



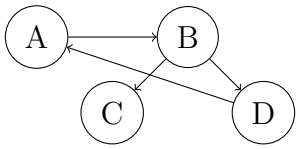
Fonte: Autor.

Seja A coleta de dados, B detecção de *concept drift*, C o retreino e D o reagendamento.

Quando o grafo é iniciado, A é o ponto de partida. Os dados serão coletados e, se a coleta for bem sucedida, B realiza a detecção de *concept drift*. Se houver *concept drift*, o fluxo seguirá para C; se não houver, segue para D, reagendando a tarefa.

Um dos conceitos importantes de DAGs é a sua propriedade acíclica, não permitindo grafos como mostrado na Figura 6. Essa propriedade garante que o *workflow* não execute o mesmo nó duas vezes na mesma tarefa.

Figura 6: Grafo cíclico



Fonte: Autor.

2.3.3 Criação e Manutenção de DAGs

De acordo com o manual Apache (2023), existem duas maneiras de instanciar uma DAG: como um objeto ou como Python *decorator*. Neste caso, a metodologia utilizada será a instanciação como objeto podendo ser visto na Listing 1. Vale se atentar que tal decisão foi tomada de maneira totalmente arbitraria, ambas as maneiras de instanciação de DAGs são intercambiáveis.

```
from airflow import DAG
from airflow.operators.bash import BashOperator
import pendulum
with DAG(
    'exemplo_dag',
    description='Uma DAG de exemplo',
    schedule='@daily',
    start_date=pendulum.now(),
```

```
    catchup=False,
    tags=['exemplo'],
    ) as dag:
        task1 = BashOperator(
            task_id="PrimeiraTask",
            bash_command="echo task1",
        )
        task2 = BashOperator(
            task_id="SegundaTask",
            bash_command="echo task2",
        )
task2 >> task1
```

Listing 1: Exemplo de DAG

No código apresentado, há a instanciação de uma DAG chamada de *'dag'* com um intervalo de execução diário. Há também a criação de duas tarefas simples que exibem no terminal *task1* e *task2*. Na última linha, foi declarada a ordem de execução onde é indicado que a *task2* deve ser executada antes da *task1*. Dessa forma a saída e registro da *task2* podem ser observados na Listing 2. Isso indica o sucesso na execução da *task2* e permite que a *task1* seja executada em seguida.

```
[2023-07-14, 13:24:37 UTC] {subprocess.py:75} INFO - Running command: ['/bin
   /bash', '-c', 'echo task2']
[2023-07-14, 13:24:37 UTC] {subprocess.py:86} INFO - Output:
[2023-07-14, 13:24:37 UTC] {subprocess.py:93} INFO - task2
[2023-07-14, 13:24:37 UTC] {subprocess.py:97} INFO - Command exited with
   return code 0
```

Listing 2: Saída Task2

2.3.4 Criação e Manutenção de Tarefas

Tarefas são a parte principal de uma DAG e podem ser adicionadas de diversas formas, como em código Python, execução de *shell script* e sensores externos. Neste subtópico, será abordada a criação e execução de tarefas.

As tarefas podem ser definidas como um nó de um grafo e devem ser escritas em formato de código Python. Cada tarefa pode realizar uma ação em determinado ou executar um processo, contribuindo para o fluxo de trabalho. A criação de tarefas pode ser exemplificada na Listing 3.

```
... #  DAG
def sayOlaMundo():
    print('Ola Mundo! Eu sou uma tarefa')

taskDummy = DummyOperator(task_id='dummy')
```

```
taskPython = PythonOperator(task_id='sayOlaMundo',
             python_callabe=sayOlaMundo)
taskShell = BashOperator(task_id='shell',
            bash_command='echo Ola do Bash')
```

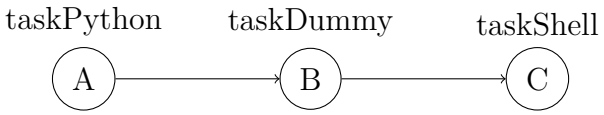
Listing 3: Criação de Tarefas

Foram criadas três tarefas:

- taskDummy: Não tem função específica, tendo caráter apenas demonstrativo.
- taskPython: Chama a função *sayOlaMundo* definida na linha 2, que imprime a mensagem 'Ola Mundo! Eu sou uma tarefa'.
- taskShell: Utiliza o terminal *Bash* para execução de comandos, incluindo *scripts shell*. Nesse caso, o comando é `echo` com o texto impresso no `stdout` *'Ola do Bash'*.

Supondo que a execução dessas tarefas seja na ordem taskPython, taskDummy e, por fim, taskShell, seu grafo poderia ser representado como na Figura 7

Figura 7: Grafo das Tarefas



Fonte: Autor.

Dessa forma a codificação definida pelo grafo, com a declaração de uma DAG pode ser vista na Listing 4.

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator
from airflow.operators.dummy_operator import DummyOperator

import pendulum
with DAG(
    'exemplo_dag',
    description='Dag para tarefas',
    schedule='@daily',
    start_date=pendulum.now(),
    catchup=False,
    tags=['exemplo'],
    ) as dag:
    def sayOlaMundo():
        print('Ola Mundo! Eu sou uma tarefa')

```

```
18     taskDummy = DummyOperator(task_id='dummy')
19     taskPython = PythonOperator(task_id='sayOlaMundo',
20                 python_callable=sayOlaMundo)
21     taskShell = BashOperator(task_id='shell',
22                 bash_command='echo Ola do Bash')
23
24 taskPython >> taskDummy >> taskShell
```

Listing 4: Tarefa completa defina pelo grafo.

No código representado pela Listing 4, as tarefas taskDummy, taskPython e taskShell são criadas e então conectadas utilizando o operador '>>' que por sua vez define a ordem de execução correta.

2.4 API (Application Programming Interface)

Uma API pode ser definida como um conjunto de regras e protocolos que permitem diferentes softwares se comunicarem entre si. Define metodologicamente estruturas e convenções que desenvolvedores podem utilizar para interagir com um sistema. Atua como uma camada intermediária, permitindo que aplicações padronizem a sua comunicação e troca de informações (IBM, 2023).

2.4.1 Tipos de API

Cada tipo de API possui uma finalidade e funcionalidade, os tipos mais encontrados são:

- Web APIs: Utilizam protocolos Web para inter-comunicação, como HTTP e HTTPs. Permitem que aplicativos com acesso a internet troquem informações por meio de JSON ou XML (MOZILLA, 2023b).
- RESTful APIs: Baseados na arquitetura REST (Representational State Transfer). Também utilizam de metodos HTTP, tais como GET, PUT, POST e DELETE especificados na Tabela 1 (REDHAT, 2023a).
- SOAP APIs: Utilizam o protocolo SOAP (Simple Object Acess Protocol). Oferece suporte a transações e segurança de processos, tornando, na maioria da vezes, mais complexa que APIs RESTful (REDHAT, 2023b).

Tabela 1: Métodos HTTP e suas descrições

Método HTTP	**Descrição**
GET	Recupera dados de uma fonte
POST	Envia dados para uma fonte.
PUT	Atualiza completamente uma fonte de dados.
PATCH	Atualiza parcialmente dados de uma fonte de dados.
DELETE	Remove um dado em específico.
HEAD	Recebe apenas a cabeça da resposta e não o corpo.
OPTIONS	Retorna os métodos HTTP disponíveis.

Fonte: Mozilla (2023a).

2.4.2 Vantagens da utilização de API

Em vários casos APIs podem ser utilizadas para evitar a criação de códigos que executam as mesmas tarefas (IBM, 2023). Dentre essa e outras vantagens podemos citar:

- Reutilização de Código: Permite que desenvolvedores foquem em novas funcionalidades, evitando a repetição de código.
- Escalabilidade: APIs permitem que sistemas sejam facilmente escalados, adicionando novas funcionalidades e novas demandas.
- Integração: Dois sistemas programados em linguagens incompatíveis podem ser ligados por meio de uma API. Tornando a operabilidade do sistema homogênea mesmo com sistemas heterogêneos.

3 Metodologia

Tendo em vista que as informações do DNS podem ser consideradas sensíveis, especialmente por conterem diversos dados que possibilitam a identificação de uma pessoa, a verificação de *concept drift* deve ser realizada localmente, e não em um ambiente de produção. Portanto, fica evidente a necessidade de criar um algoritmo que possa gerar e simular conjuntos de dados semelhantes aos reais que serão analisados durante a implantação.

3.1 Conjunto de dados artificial

Conforme mencionado em Silva (2022), as informações que fazem parte do conjunto de dados atualmente utilizadas em produção estão descritas na Tabela 2

Além disso, todas as strings foram convertidas de maneira lógica para números inteiros. Nesse processo, o tipo *qtype* foi transformado em um número de 1 a 3 (correspondente

Tabela 2: Campos do conjunto de dados

Feature	**Descrição**	**Tipo**
do_epp_provider	Provedor EPP	Booleano
nb_days_until_collect	Número de dias do registro até a primeira consulta	Inteiro
ttl	Time to Live	Inteiro
ipv	Versão do IP	Inteiro
prot	Protocolo TCP ou UDP	Inteiro
srcp	Porta de origem	Inteiro
aa	*Authoritative Answer*	Booleano
cd	Checking Disable	Booleano
ancount	*Answer Count*	Inteiro
arcount	*Additional Information Count*	Inteiro
nscount	*Authority Count*	Inteiro
rcode	*Responde Code*	Inteiro
qtype	Tipo de *Quarry*	String
country	País	String
asn	*Autonomous System Number*	Inteiro
labels	Títulos	Booleano
res_len	Tamanho da resposta	Inteiro

Fonte: Silva (2022).

à quantidade existente de *quarys*), enquanto a variável *country* foi representada numericamente, variando de 1 a 195, correspondendo ao número total de países existentes.

Após a conversão, todos os campos se tornaram numéricos e passíveis de geração aleatória. Para a criação dos conjuntos de dados, optou-se por uma separação arbitrária em dois *clusters*, onde os valores foram divididos aleatoriamente ao meio. Esses clusters podem ser visualizados na Tabela 3

Alguns pontos foram cruciais para a separação dos valores:

- Overlap aleatório de campos
- Exclusão aleatória de valores
- Valor fixo de campos

Cada um desses pontos permite gerar conjuntos de dados imperfeitos e, consequentemente, mais próximos do real. Dessa forma, é possível verificar se o detector de *concept drift* consegue permanecer eficaz mesmo em condições mais complexas.

O algoritmo selecionado para treinamento foi o mesmo utilizado por Silva (2022), *LGBMClassifier*, provado eficaz para dados similares aos gerados de forma artificial.

Tabela 3: Separação de dataset

Feature	**Cluster 1**	**Cluster 2**
do_epp_provider	Verdadeiro ou falso	Verdadeiro ou falso
nb_days_until_collect	1 a 30	10 a 365
ttl	800 a 1000	1 a 500
ipv	4	4 ou 6
prot	1	1 ou 2
srcp	50000 a 65535	1 a 50000
aa	Verdadeiro	Verdadeiro ou falso
cd	Falso	Verdadeiro ou falso
ancount	50 a 100	0 a 50
arcount	50 a 100	0 a 50
nscount	50 a 100	0 a 50
rcode	0 a 1	0 a 15
qtype	1	1 a 3
country	1 a 97	98 a 195
asn	1 a 5000	5001 a 10000
labels	Verdadeiro ou falso	Verdadeiro ou falso
res_len	800 a 1000	1 a 800

Fonte: Autor.

3.2 Introdução de concept drift

Com o algoritmo para a criação de conjuntos de dados, é possível introduzir variações de conceitos que, teoricamente, farão o classificador errar. Para isso, serão criadas variações e alterações de informação diretamente no conjunto de dados, duas técnicas foram empregadas para a alteração dos dados: Alteração de conceito abrupta e incremental. A diferença das duas é o enfoque na quantidade de mudança com o tempo.

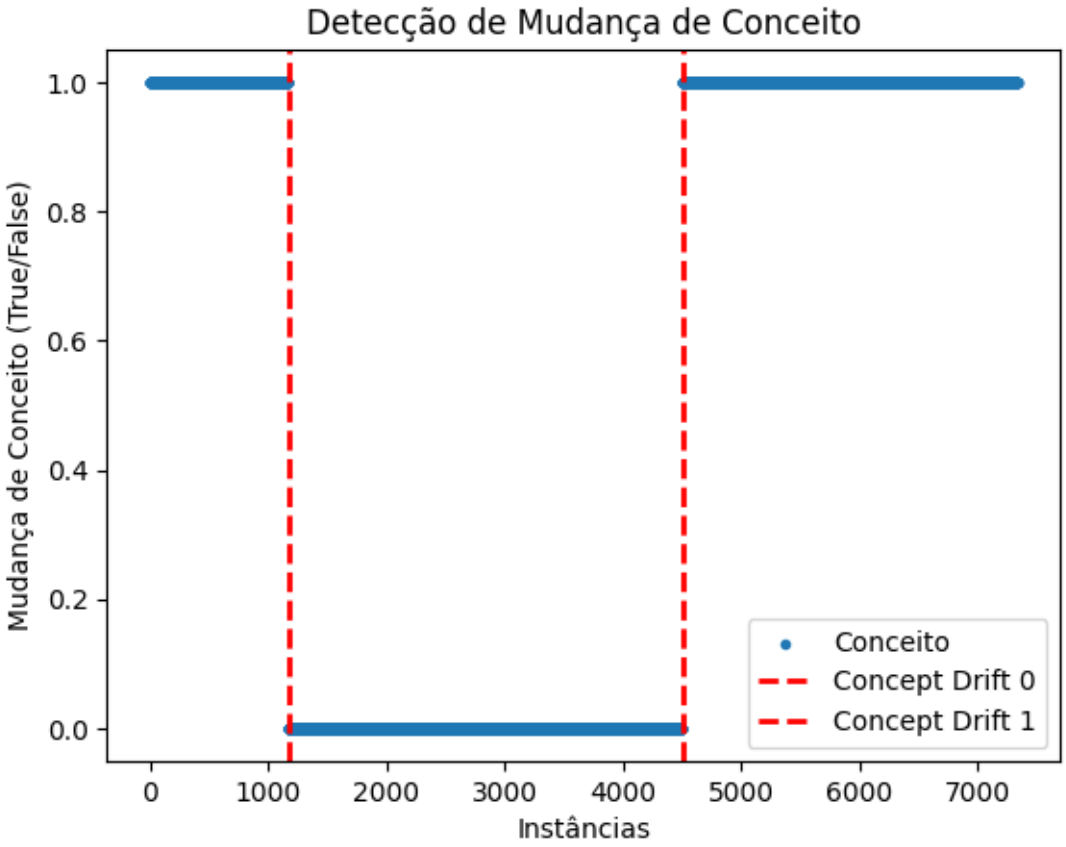
A alteração de conceito abrupta consiste em alterar os rótulos de todos os dados do conjunto de dados. Por exemplo, um conjunto de dados que classifica DNS como fraudulentos pode ser alterado para classificar todos os DNS como legítimos. Enquanto a alteração de conceito incremental consiste em alterar os dados do conjunto de dados de forma gradual. A título de exemplo, um conjunto de dados que classifica DNS como fraudulentos pode ser alterado para classificar DNS com maior probabilidade de serem legítimos com o decorrer do tempo.

Quanto ao algoritmo de detecção de *concept drift*, optou-se pelo KSwin, conhecido por sua capacidade de identificar mudanças de conceito ao longo de grandes períodos, mesmo em cenários com mudanças graduais (RIVER, 2023).

Após o treinamento artificial do classificador, o modelo é utilizado para fazer previsões sobre dados novos e rotulados. Essas previsões são posteriormente enviadas para o KSwin, que analisa as diferenças entre as previsões do classificador e os rótulos reais dos dados.

Caso haja um *concept drift*, o KSwin armazena as informações em um array, que é então representado graficamente em um gráfico de pontos, como na Figura 8. O gráfico representa cada ponto de dados como um ponto no gráfico. O eixo X representa o tempo e o eixo Y representa a distância entre as previsões do classificador e os rótulos reais dos dados. A linha vermelha indica a mudança de conceito.

Figura 8: Exemplo de gráfico



Fonte: Autor.

3.3 Fluxo de trabalho em Airflow

Certos pontos são de extrema importância nesta estrutura. O sistema de classificação, como descrito em Silva (2022), demonstrou sua eficácia ao empregar diversos tipos de algoritmos classificadores. No contexto da verificação de *concept drift*, o tipo de classificador utilizado não é crucial; o que importa são os rótulos previstos e os rótulos reais. Portanto, apenas essas informações devem ser enviadas para o *Apache Airflow*. Para a produção, um JSON estruturado deve ser transmitido, contendo as informações presentes na Listing 5.

Nessa estrutura, `date` representa a data da predição, com ano, mês e dia em ordem. `p_label` é o rótulo previsto e `a_label` é o rótulo real.

```
[{
date: "20231030",
p_label: "True",
a_label: "False"
}]
```

Listing 5: Exemplo de JSON.

Esses dados, serializados em JSON, serão constantemente monitorados e salvos em um arquivo chamado 'labels.json'. Quando o momento de execução agendado chegar, a DAG será acionada para analisar os dados recebidos. Outra abordagem possível é processar os dados imediatamente após o recebimento. Independentemente do método escolhido, o *Apache Airflow* é capaz de gerenciar esses fluxos de dados. Qualquer informação datada há mais de um ano será automaticamente descartada.

Mensalmente, os dados dos rótulos serão analisados pelo algoritmo KSwin. Além disso, se houver indícios de *concept drift*, notificações por e-mail serão enviadas a todos os usuários cadastrados. Adicionalmente, um gráfico de dispersão será incluído como parte do relatório mensal.

No caso de detectar-se *concept drift*, será solicitado um novo treinamento do modelo. Um conjunto de dados atualizado será preparado para permitir que o modelo de aprendizado se ajuste às mudanças identificadas, garantindo assim a precisão contínua das predições. Dessa forma, a estruturação em forma de grafo acíclico pode ser definida no formato representado na Figura 9, assim como a implementação referenciada na Listing 6.

```
#... Imports
with DAG(
    'verify_drift',
    description='O evento de verificação de concept drift será realizado
   mensalmente. The concept drift verification event will happen monthly',
    schedule='@monthly',
    start_date=pendulum.now(),
    catchup=False,
    tags=['drift verification'],
    ) as dag:
        load_task = PythonOperator(
            task_id="load_labels",
            python_callabe=load_label,
        )
        detector_task = PythonOperator(
            task_id="verify_drift",
            python_callabe=verify_drift,
        )
        graph_task = PythonOperator(
            task_id="send_graph",
            python_callabe=drift_ok,
```

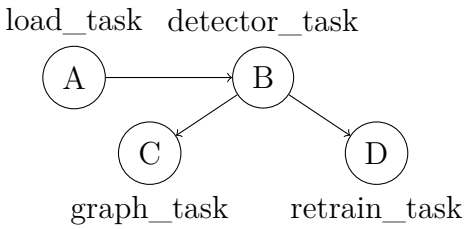
```
        )
        retrain_task = PythonOperator(
            task_id="retrain",
            python_callabe=call_retrain,
        )
        branch = BranchPythonOperator(
            task_id="branch",
            python_callable=lambda f: "retrain" if f == True else "
    send_graph",
            op_kwargs={"f": drift_detected},
            dag=dag)

load_task >> detector_task >> [graph_task,retrain_task]

#... Def's
```

Listing 6: Fluxo de trabalho

Figura 9: Representação de Fluxo



Fonte: Autor.

Neste fluxo, inicialmente, a tarefa de carregamento (load_task) é acionada para extrair as informações contidas no arquivo 'labels.json'. Em seguida, começa a tarefa do detector KSwin (detector_task), cuja resposta determinará o curso subsequente dos eventos. Dependendo da avaliação do detector, o sistema pode seguir duas rotas distintas:

- **Tarefa de Retreino**: Se o detector KSwin identificar mudanças significativas nos dados, a DAG prossegue para a tarefa de retreino (retrain_task). Esta etapa é crucial, pois implica que o modelo atual contém *concept drift*, e deve ser retreinado com uma nova base de dados atualizada.
- **Tarefa de construção do gráfico**: Se o detector KSwin não identificar nenhum desvio significativo nos dados, a DAG encaminha-se naturalmente para a tarefa de construção do gráfico (graph_task). Este gráfico oferece uma visualização ampla do encaminhamento de desvio, sendo essencial para a tomada de decisões futuras.

3.4 Testes e Resultados

A geração de conjuntos de dados, como mencionado anteriormente, ocorre de forma aleatória, enquanto a introdução do *concept drift* é realizada ao selecionar a área onde ocorrerá a mudança. A estrutura pode ser exemplificada como na Listing 7.

```
dataframe = criação_aleatória_de_dados(tamanho)
dataframe.append(criação_concept_drift(posição, tamanho))
dataframe.append(criação_aleatória_de_dados(tamanho))
```

Listing 7: Pseudocódigo de criação de dados

Dessa forma, um conjunto de dados contendo desvios de conceito é gerado e armazenado. Isso permite testar a eficácia do modelo selecionado para detectar *concept drift*. Antes de testar o *concept drift*, é necessário criar e treinar o classificador. O processo envolve a separação dos dados em conjuntos de treinamento e teste, conforme exemplificado no pseudocódigo apresentado na Listing 8

```
y = dataframe[rotulos]
X = dataframe[demais_dados]
X_treino, y_treino, X_teste, y_teste = separação_treino_teste(X, y)
classificador.treinar(X_treino, y_treino)
y_predicao = classificador.predizer(X_teste)
```

Listing 8: Pseudocódigo de classificador

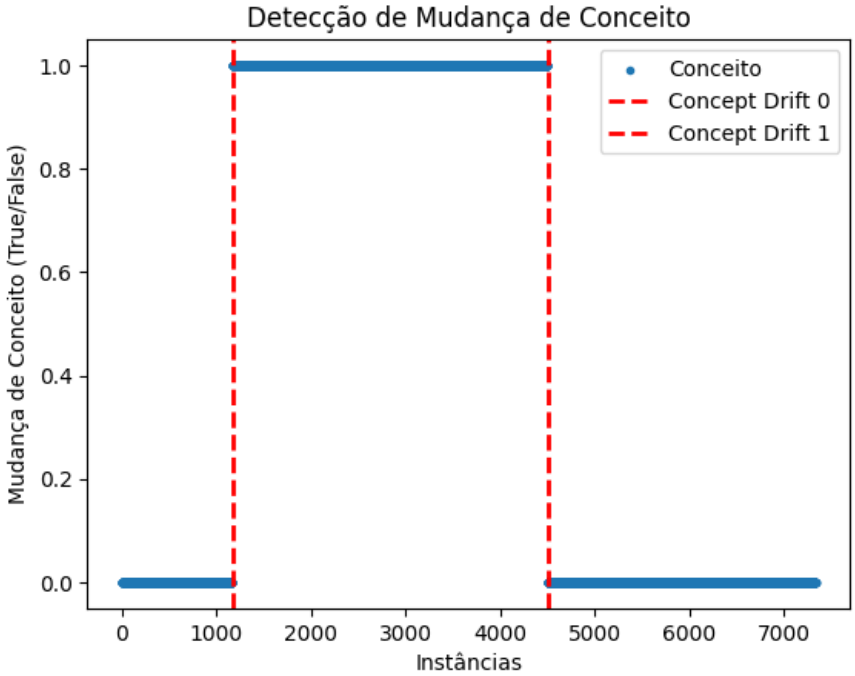
Após o treinamento, o classificador está pronto para servir como base de teste para o detector de *concept drift*. O detector compara as predições do modelo com os dados reais para verificar se há desvios de conceito. Como o classificador apenas precisa ser treinado uma vez e está otimizado conforme o estado atual de Silva (2022), a parte classificatória será utilizado como um conceito para testar o classificador em uma base de dados que poderia ser analisada em consultas diárias de DNS passivo. Para testar o detector, pode-se utilizar um código semelhante ao pseudocódigo presente na Listing 9.

```
escala = valor_atual[posição] igual a valor_real[posição]
detector.adicionar_elemento(escala)
se detector.detecta()
    então salva(posição)
```

Listing 9: Pseudocódigo Detector de Concept Drift

Assim, o detector avalia os valores recebidos e salva a posição se houver desvio de conceito. Esse método permite identificar mudanças significativas nos dados, possibilitando ajustes no modelo ou nos processos que dependem desses dados. Após gerar, treinar e verificar os dados, foi identificado que o modelo treinado sofreu *concept drift* nas posições 1180 e 4513, conforme mostrado na Figura 10. Essa figura revela uma variação abrupta no conceito dos dados, sendo o código responsável pela geração desse conjunto de dados referenciado na Listing 10.

Figura 10: Resultado da execução



Fonte: Autor.

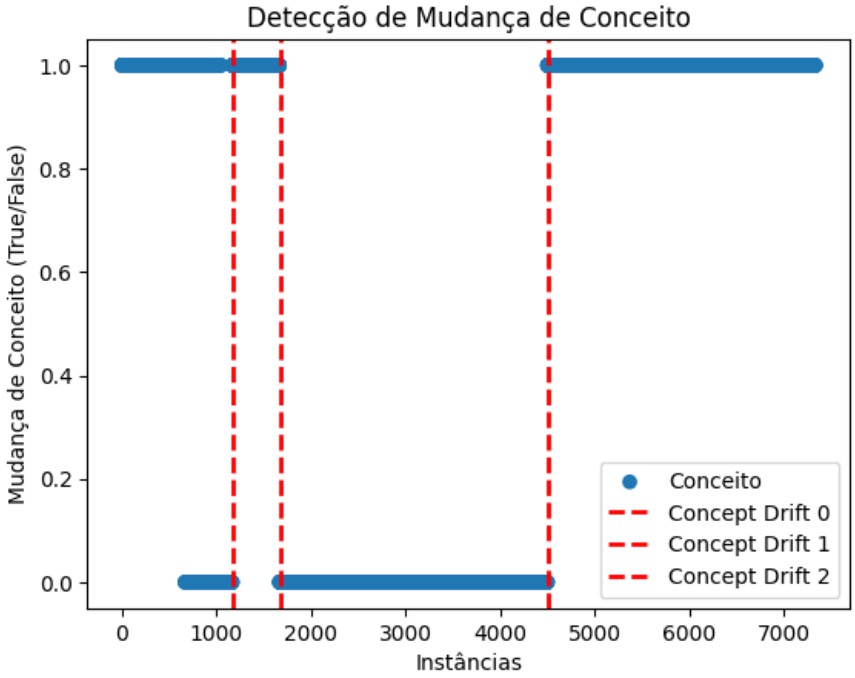
```
data = generate_data(int(float(10000)/1.5))
data = pd.concat([data, generate_data(int(float(10000)*0.1), True)],
    ignore_index=True)
data = pd.concat([data, generate_data(int(float(10000)/1.5) - int(float
    (10000)*0.1))], ignore_index=True)
```

Listing 10: Geração de dados abrupta

É interessante notar que a região onde ocorre o *drift* é exatamente a mesma onde o segundo conjunto de dados foi gerado (o segundo conjunto contém a *flag* de *concept drift*). Isso prova que o detector é capaz de identificar *drift* abrupto em conjuntos de dados muito próximos do mundo real.

Além do tipo de *drift* abrupto, existe também a geração de *drift* incremental, onde o conceito muda gradualmente ao longo do tempo, ocorrendo alterações rápidas, o que possibilita a detecção de vários *drift* em períodos de tempo próximos, como na Figura 11. No modelo incremental, a probabilidade de ocorrer um *drift* aumenta à medida que o tempo avança. Para monitorar essa probabilidade, uma variável é empregada como referenciado na Listing 11.

Figura 11: Resultado da execução incremental



Fonte: Autor.

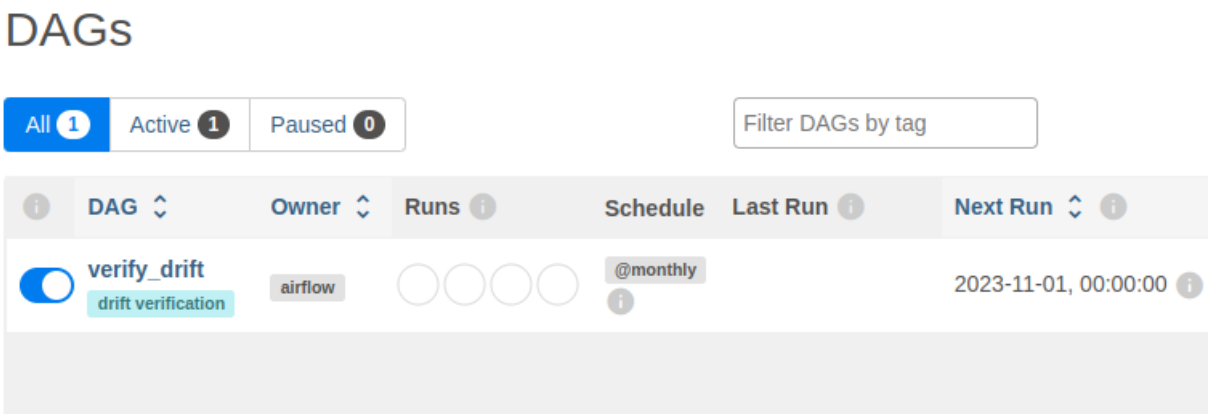
```
data = generate_data(int(float(10000)/1.5))
data = pd.concat([data,generate_data(int(float(10000)*0.1),drift=True,cdr
    =0.2)],ignore_index=True)
data = pd.concat([data,generate_data(int(float(10000)/1.5)-int(float(10000)
    *0.1))],ignore_index=True)
```

Listing 11: Geração de dados incremental

Aqui, ”cdr”representa a Taxa de Mudança de Conceito (*concept drift rate*), indicando o aumento na taxa de mudança de conceito a cada iteração do loop.

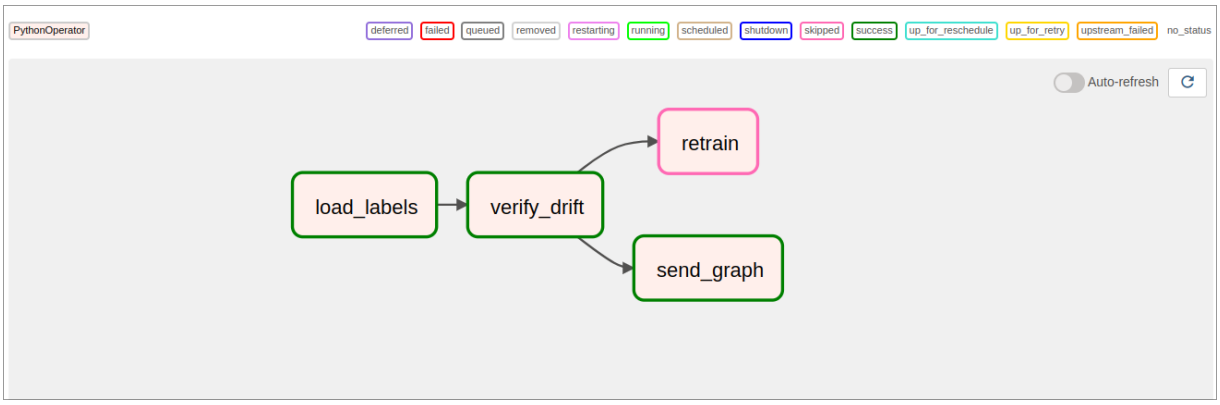
Através desse processo, é possível testar o fluxo de trabalho que foi apresentado anteriormente. A página inicial do *Apache Airflow*, como mostrado na Figura 12, é o ponto de

Figura 12: Vista de usuário



Fonte: Autor.

Figura 13: Vista do usuário em DAG



Fonte: Autor.

partida. Nesta página, é permitido ativar ou desativar a *DAG* manualmente, dependendo das necessidades do momento. Além disso, é possível identificar quem a criou, visualizar os resultados anteriores de execução (que, neste caso, estão vazios), entender a periodicidade da execução e verificar quando será a próxima execução agendada.

Cada *DAG* pode ser acessada individualmente, proporcionando uma visão detalhada, como mostrado na Figura 13. O grafo acíclico é plotado automaticamente pelo *Apache Airflow*. Uma característica interessante é a capacidade de ultrapassar o *Scheduler* e forçar a execução do *workflow* imediatamente. Isso significa que é possível testar o fluxo a qualquer momento, garantindo flexibilidade no processo de teste. Ao forçar a execução, os detalhes da execução de cada um dos nós podem ser encontrados na aba de *Logs*. Um exemplo desse registro é apresentado na Listing 12. Neste exemplo, não houve falhas e a tarefa de gerar gráficos foi bem-sucedida. Esta visualização detalhada oferece uma compreensão completa das operações executadas em cada nó.

```
1 {standard_task_runner.py:85} INFO - Job 94: Subtask send_graph
```

```
{logging_mixin.py:150} INFO - INFO: NO DRIFT DETECTED, GENERATING GRAPH!
{python.py:183} INFO - Done. Returned value was: None
{taskinstance.py:1350} INFO - Marking task as SUCCESS. dag_id=verify_drift,
    task_id=send_graph
{local_task_job_runner.py:225} INFO - Task exited with return code 0
```

Listing 12: Execução de dag.

Em outra situação, caso ocorressem falhas, seria possível acionar o processo de retreinamento automatizado. Isso seria feito enviando um novo conjunto de dados com informações atualizadas por meio da API mencionada em Gardini (2022), a fim de evitar falsos positivos no algoritmo atual de detecção de abuso de DNS.

Dessa forma, torna-se evidente a integração eficaz do verificador de *drift* com o *workflow* do *Apache Airflow*. Esse sistema oferece não apenas uma visão clara do processo, mas também a capacidade de testar e validar o fluxo de trabalho de forma contínua e precisa.

Com isso, é possível realizar a comparação de desempenho de previsão com conjuntos de dados que apresentam *drift*, utilizando o método empregado por Silva (2022) em comparação com o método utilizado na monografia. No momento em que o *concept drift* é detectado, o modelo empregado continua a seguir o mesmo padrão de treinamento anterior, conforme representado pela Figura 8, independentemente de haver ou não mudança de conceito, resultando na realização da predição da mesma forma.

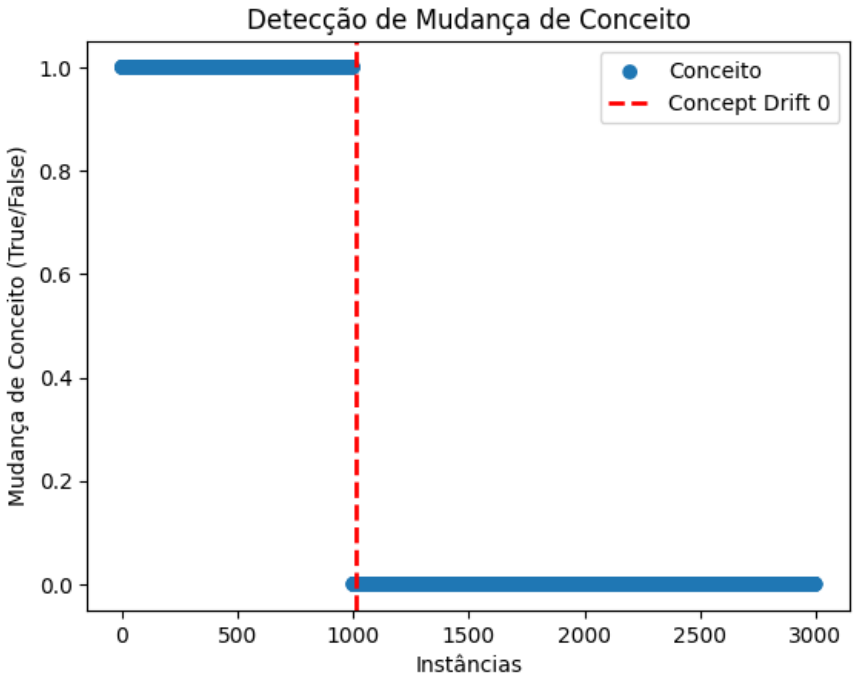
Ao adotar a nova metodologia, espera-se obter resultados diferentes, como ilustrado na Figura 14, onde a mudança de conceito ocorre e, posteriormente, o modelo é retreinado para que, nas próximas avaliações, ocorra algo semelhante ao mostrado na Figura 15.

4 Conclusão

A crescente notoriedade das pesquisas relacionadas ao abuso nos sistemas de nomes de domínio (DNS) é um claro sintoma da evolução das constantes ameaças digitais. Nesta monografia, esse domínio foi explorado utilizando conceitos de inteligência artificial e detecção de *concept drift* para enfrentar um desafio complexo: manter modelos de segurança atualizados e adaptáveis a um mundo onde conceitos e padrões mudam de forma abrupta e imprevisível.

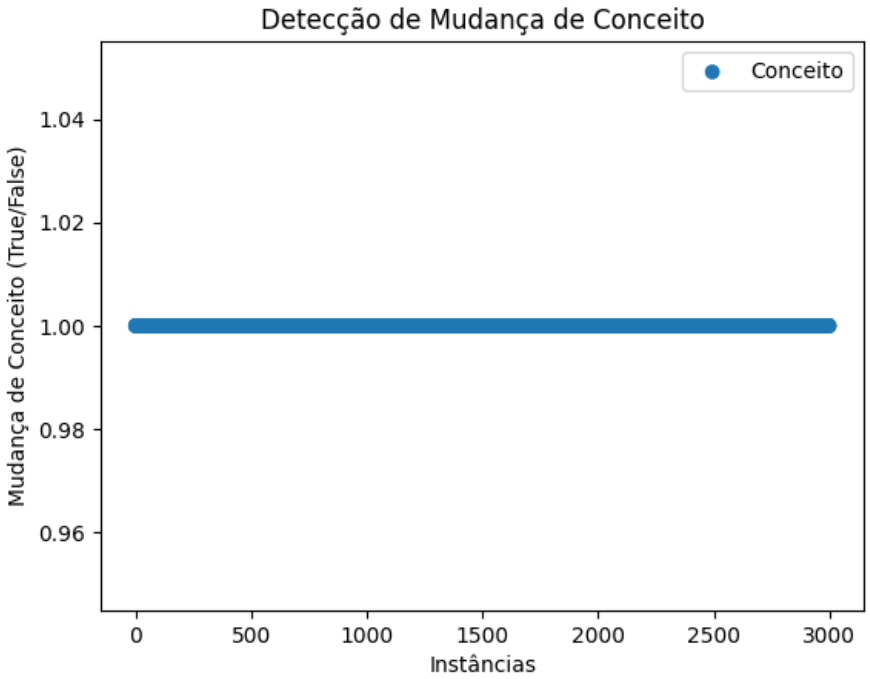
Este estudo se torna necessário especialmente no contexto da proteção do DNS, essencialmente provido e operado pela NIC.br. A necessidade de evoluir além dos modelos atualmente empregados de forma estática é evidente, especialmente quando consideramos eventos globais imprevisíveis, como pandemias e crises políticas, que têm o grande poder de alterar drasticamente comportamentos na internet e, por consequência, padrões

Figura 14: Nova metodologia detectando variação de conceito.



Fonte: Autor.

Figura 15: Nova metodologia apoś retreino.



Fonte: Autor.

de ataques e abusos no DNS. Garantir que os modelos de segurança estejam um passo à frente das mudanças torna-se crucial para proteger sistemas críticos contra DNS *phishing* e outros tipos de ataques.

Os resultados apresentados pelos gráficos (Figura 10 e Figura 11) indicam que o algoritmo KSwin selecionado para a detecção de *concept drift* foi altamente eficaz. Ao compararmos o modelo incremental desenvolvido com o modelo estático existente, fica claro que a abordagem dinâmica, onde são previstas alterações de conceito e a consistência dos dados é verificada continuamente, supera as abordagens estáticas.

A implementação bem-sucedida em um ambiente parece não apenas viável, mas também benéfica para as organizações que dependem de DNS seguros. Ao adotar uma mentalidade altamente adaptativa e integrar técnicas de detecção de *concept drift*, as organizações que fazem o controle de DNS confiável conseguem enfrentar novos tipos de ataques com maior facilidade.

Apesar de este estudo representar um avanço na segurança dos sistemas DNS, sempre haverá espaço para futuras melhorias e pesquisas. Investigar pequenas alterações no *concept drift* em cenários alternativos pode aprimorar ainda mais a evolução do algoritmo atual. Dessa forma, estudos que aprimorem essas interações são validos candidatos a sucessão do projeto e podem servir como avanço para novas pesquisas no ramo de segurança em DNS.

Em última análise, a monografia não apenas permite a exploração técnica, mas também destaca a constante evolução na proteção contra ataques no DNS. À medida que padrões mudam, as estratégias de segurança devem evoluir para acompanhar, garantindo que os sistemas críticos possam continuar protegendo todos que utilizam a internet.

Referências

APACHE. *What is airflow?* Apache, 2023. Acessado em 12-06-2023. Disponível em: <https://airflow.apache.org/docs/apache-airflow/stable/index.html>.

BAYRAM, F.; AHMED, B. S.; KASSLER, A. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, v. 245, p. 108632, 2022. ISSN 0950-7051.

CISCO. *Passive DNS*. 2022. <https://docs.umbrella.com/investigate/docs/passive-dns#:~:text=About%20Passive%20DNS&text=With%20passive%20DNS%20data%2C%20you,records%20for%20a%20malicious%20site>. Acessado em 12-06-2023.

GARDINI. Victor fernandes automatização de processos de machine learning do framework dns para a detecção de domínios maliciosos. *Universidade Estadual Paulista (Unesp)*, 2022.

GONÇALVES, P. M. et al. A comparative study on concept drift detectors. *Expert Systems with Applications*, v. 41, n. 18, p. 8144–8156, 2014. ISSN 0957-4174.

IBM. *API Concepts*. 2023. Acessado em 14-06-2023. Disponível em: <https://www.ibm.com/docs/en/i/7.1/topic/interfaces-api-concepts>.

ICANN. *Acronyms and Terms*. 2023. Acessado em 13-06-2023. Disponível em: <https://www.icann.org/en/icann-acronyms-and-terms/domain-name-system-abuse-en>.

KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-down Approach*. [S.l.: s.n.], 2017.

MOZILLA. *HTTP Request Methods*. 2023. Acessado em: 18-07-2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>.

MOZILLA. *Introduction to Web APIs*. 2023. Acessado em: 18-07-2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction>.

NIC.BR. *NIC.br Passa a marca de Cinco Milhões de Domínios Registrados*. 2022.

REDHAT. *What is a REST API?* 2023. Acessado em: 18-07-2023. Disponível em: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.

REDHAT. *What is the difference between SOAP and REST?* 2023. Acessado em: 18-07-2023. Disponível em: <https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest>.

RFC. *Domain names - concepts and facilities*. [S.l.]: RFC Editor, 1987. RFC 1034. (Request for Comments, 1034).

RIVER. *KSWIN - river*. 2023. <https://riverml.xyz/0.18.0/api/drift/KSWIN>. Accessed: 2023-10-10.

SILVA, L. M. d. Early identification of abused domains in tld through passive dns applying machine learning techniques. *International Journal of Communication Networks and Information Security*, v. 14, p. 76–85,, 2022. N.

TOGBE, M. U. et al. Anomalies detection using isolation in concept-drifting data streams. *Computers*, MDPI AG, v. 10, n. 1, p. 13, jan. 2021.