Trabalho 2 da disciplina de Gerência de Redes - Gerente de contabilidade

Enrico D. Pizzol[1], **Cássio Entrudo**[1], **Giovani D. Silva**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)

{enrico.pizzol,giovani.silva,cmentrudo}@inf.ufrgs.br

Resumo. *O trabalho proposto teve como objetivo desenvolver uma ferramenta gráfica capaz de realizar três tarefas de gerenciamento de contabilização. A biblioteca SNMPv3 foi utilizada para a elaboração da tarefa. A biblioteca PySimpleGui foi utilizada para a elaboração da interface gráfica devido à sua facilidade de uso. Em sintese, a ferramente elaborada pelo grupo serve como um painel de monitoramente, onde o gerente consegue tomar conhecimento de um série de informaçõees.*

1. Parte 1 - Resumo da Etapa 1

1.1. Resumo

O trabalho proposto teve como objetivo desenvolver uma ferramenta gráfica capaz de realizar três tarefas de gerenciamento de contabilização. A biblioteca SNMPv3 foi utilizada para a elaboração da tarefa. A biblioteca PySimpleGui foi utilizada para a elaboração da interface gráfica devido à sua facilidade de uso. Em sintese, a ferramente elaborada pelo grupo serve como um painel de monitoramente, onde o gerente consegue tomar conhecimento de um série de informaçõees.

1.2. Implementação do código

O código foi desenvolvido em Python-3.6, utilizando-se das bibliotecas PySimpleGUI(para interface gráfica) e easysnmp, para o SNMNP. Junto ao código está um arquivo requirements.txt, que pode ser utilizado para instalar as bibliotecas necessárias na maquina, com os comandos a seguir: A aplicação desenvolidada é extremamente simples, contendo apenas um **launcher** e uma classe **App** para quem delegamos a criação da interface. A classe **App** também é responsável por escutar os eventos e atualizar os valores na tela.

1.3. Metodologia de Análise

A metodologia de analise, além do visual, baseia-se na função de verificação de erros, ela recebe o valor da variável e um parâmetro de limite, caso passe do limite, avisa o gerente e pergunta se quer fechar a aplicação.

1.4. Execução do programa

Abaixo, apresentamos um print de tela com a execução do programa, na qual uma janela do youtube está aberta.

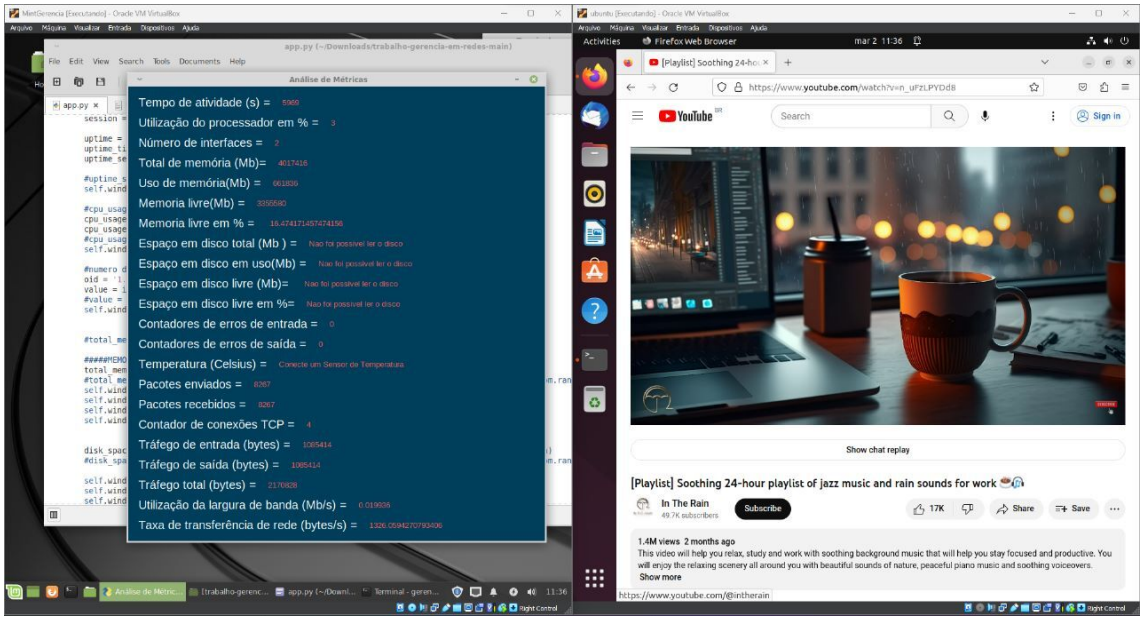

Figura 1. Execução do código na máquina gerente

2. Parte 2 - Implementação de Machine Learning para Gerenciamento de Contabilidade de Rede

Neste trabalho, aplicamos algoritmos de aprendizado de máquina não supervisionado para monitorar e gerenciar a contabilidade da rede, visando identificar anomalias, alterações nos padrões e possíveis erros indesejados. Esses algoritmos exploram os dados e identificam padrões, permitindo assim a detecção de eventos incomuns. Descrevemos brevemente quatro algoritmos utilizados no contexto do gerenciamento de contabilidade de rede, bem como suas respectivas implementações.

2.1. Definição do modelo de aprendizado de máquina

Antes de iniciar o processo de treinamento do modelo de detecção de anomalias, foi necessário definir o modelo de aprendizado de máquina a ser utilizado. A escolha do modelo de aprendizado de máquina depende do problema que se deseja resolver e dos dados disponíveis para treinamento. Neste projeto, optamos por utilizar a Isolation Forest, um algoritmo de aprendizado não-supervisionado baseado em árvores de decisão, devido à sua capacidade de detectar anomalias em tempo real em grandes volumes de dados.

Após a definição do modelo de aprendizado de máquina, foram identificadas as variáveis necessárias para o treinamento e seus respectivos valores por meio da variável ”variables”.

```
variables = [cpu_usage_percent,number_of_interfaces,total_memory,us
    #variables = [cpu_usage_percent, used_memory_percent, ifInErrors,
    variables = [float(i) for i in variables]
    self.anomaly_data.append(variables)
```

Para garantir que os valores das variáveis fossem tratados corretamente, foi realizada uma conversão para float, conforme pode ser visto no print anterior. Essa conversão

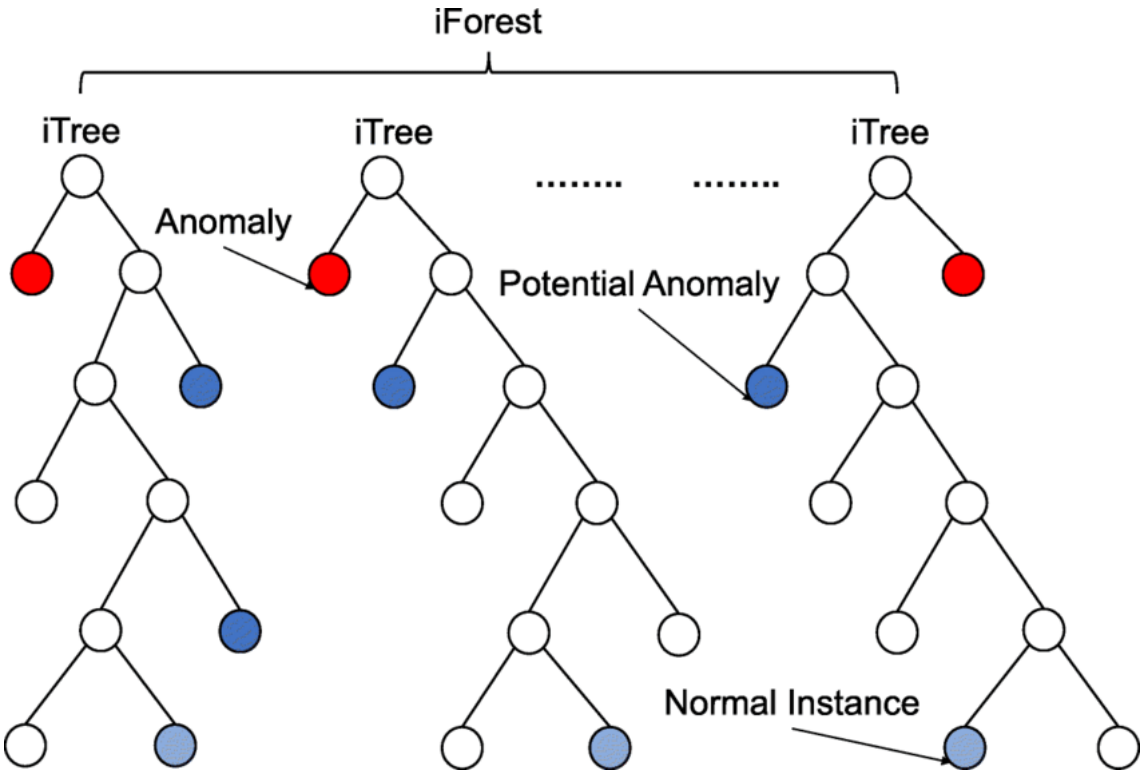

Figura 2. Ilustração de uma isolation forest,fonte:https://www.researchgate.net/figure/Isolation-Forest-learned-iForest-construction-for-toy-datasetfig1352017898

é necessária para garantir que os valores sejam interpretados corretamente pelo modelo de aprendizado de máquina e para evitar erros na detecção de anomalias.

2.2. Funcionamento da Isolation Forest

O algoritmo de detecção de anomalias utilizado neste projeto é a Isolation Forest, um algoritmo de aprendizado não-supervisionado baseado em árvores de decisão. A Isolation Forest trabalha separando as anomalias em regiões que possuem poucos pontos e, portanto, requerem menos particionamentos para serem isoladas. Dessa forma, a Isolation Forest é capaz de detectar anomalias de forma eficiente e rápida, o que é particularmente útil em ambientes de grande volume de dados em que a detecção de anomalias em tempo real é necessária.

Para detecção de anomalias, a Isolation Forest utiliza o princípio de que as anomalias são menos frequentes e, portanto, mais fáceis de isolar do que os pontos normais. Isso é feito criando-se subamostras aleatórias dos dados de entrada e dividindo essas subamostras em subconjuntos menores. A Isolation Forest então calcula o número médio de divisões necessárias para isolar cada ponto da subamostra, o que representa uma medida de sua anomalia. Quanto menos divisões forem necessárias para isolar um ponto, maior será sua anomalia.

Com base nesse princípio, a Isolation Forest é capaz de prever um valor para aquele conjunto de dados de entrada, e caso uma anomalia seja detectada, esse valor é -1 e um popup é emitido na tela do programa informando a anomalia. Esse processo é realizado de forma rápida e eficiente, permitindo a detecção de anomalias em tempo real

e possibilitando a tomada de medidas corretivas imediatas.

2.2.1. Hiperparâmetro MINDATAPOINTS

Por se tratar de um algoritmo de aprendizado não-supervisionado, a Isolation Forest não requer dados rotulados para treinamento. No entanto, é necessário definir os dados de entrada para que o modelo possa ser treinado adequadamente. Para isso, foi utilizada a variável ”MINDATAPOINTS”, que determina a quantidade mínima de pontos a serem colocados dentro do array de pontos para que o treinamento possa ser iniciado. Esse hiperparâmetro é de suma importância para a detecção de anomalias, já que quanto maior o valor de MINDATAPOINTS, mais pontos serão utilizados e mais valores ”estranhos”deverão aparecer para que uma anomalia seja detectada.

2.2.2. Código Isolation Forest

```
#No arquivo ml.py
    def train_anomaly_detector(self,data):
        #escalarizar os dados
        scaler = StandardScaler()
        data = scaler.fit_transform(data)
        #treinar o modelo
        self.model.fit(data)
        return self.model

#no arquivo app.py
prediction = 0
        if len(self.anomaly_data) >= MIN_DATA_POINTS:
            anomaly_data_np = np.array(self.anomaly_data)
            self.anomaly_model = self.ml.train_anomaly_detector(anomaly

        if self.anomaly_model is not None:

            variables = np.array(variables)
            variables_reshaped = variables.reshape(1,-1)
            prediction = self.anomaly_model.predict(variables_reshaped)
            self.predictions.append(prediction)
            #pegar o timestamp atual
            current_timestamp = datetime.now()
            self.timestamps.append(current_timestamp)

            print(prediction)
            if prediction[0] == -1:
                print("Anomalia detectada!")
                #sg.popup('Anomalia detectada!')

            #verificar a import ncia de cada atributo
```

```
            self.ml.feature_importance(self.feature_names, self.anomaly
            #self.ml.print_tree(self.anomaly_data, self.anomaly_model, s
        else:
            self.predictions.append(prediction)
```

2.3. Atributos mais importantes

Os atributos são características ou propriedades que definem um objeto ou conjunto de dados. Na análise de dados, é comum identificar quais atributos são mais importantes para determinado modelo ou algoritmo.

No caso específico da Isolation Forest, um algoritmo de detecção de anomalias em dados, é importante identificar quais são os atributos que mais contribuem para a detecção dessas anomalias. Para isso, é possível utilizar um código que busca mostrar quais features influenciam mais na árvore.

O código abaixo pode ser utilizado para identificar os 3 atributos que mais contribuem para a Isolation Forest. Essa informação pode ser útil para a interpretação dos resultados e para a melhoria do modelo. Com base nos resultados obtidos, é possível ajustar os parâmetros do modelo ou coletar novos dados que possam melhorar a detecção de anomalias.

```
def feature_importance(self,feature_names, anomaly_model, prediction):
        X = prediction
        n_features = X.shape[1]
        feature_importance = np.zeros(n_features)
        for i in range(n_features):
         x_modified = X.copy()
         x_modified[:, i] = 0
         model_modified = IsolationForest(random_state=42).fit(x_modifi
         feature_importance[i] = np.mean([tree.feature_importances_[i]

        important_features = np.argsort(feature_importance)[-3:]
        print("As 3 metricas mais importantes na deteccao de anomalias
```

```
As 3 metricas mais importantes na detecção de anomalias são: ['total_traffic', 'utilizacao_bps', 'trafego']
```

Figura 3. Resultado atributos mais importantes

2.4. Monitoramento em tempo real

Para monitorar o fluxo de entrada de informação durante o treinamento do modelo de detecção de anomalias, foi plotado um gráfico em tempo real das variáveis utilizadas no treinamento. Essa visualização permitiu uma análise mais aprofundada dos dados e facilitou a identificação de possíveis anomalias, proporcionando uma maior compreensão do comportamento dos dados ao longo do tempo.

O gráfico em tempo real das variáveis utilizadas no treinamento foi uma ferramenta importante para garantir a eficácia do modelo de detecção de anomalias. Com essa visualização, foi possível monitorar o fluxo de entrada de informação e identificar

possíveis anomalias em tempo real, possibilitando uma rápida intervenção para evitar possíveis prejuízos ou riscos.

Além disso, a plotagem do gráfico em tempo real permitiu a detecção de padrões ou tendências nos dados, o que pode ser útil na identificação de possíveis anomalias. Dessa forma, a visualização das variáveis em tempo real durante o treinamento do modelo de detecção de anomalias proporcionou uma maior compreensão dos dados e permitiu a identificação de anomalias de forma mais precisa e eficiente.

2.4.1. Código Monitoramento em tempo real

```
    def update_realtime_chart(self, data):
        data_np = np.array(data)
        time_axis = range(data_np.shape[0])

        # Atualizar os dados dos gr ficos
        for i in range(3):
            self.ax[i].clear()

        self.ax[0].set_title("Largura de banda")
        self.ax[0].plot(time_axis, data_np[:, 13], label="Largura de ba
        self.ax[1].set_title("Uso de mem ria")
        self.ax[1].plot(time_axis, data_np[:, 5], label="Uso de mem ria
        self.ax[2].set_title("Uso de CPU")
        self.ax[2].plot(time_axis, data_np[:, 1], label="Uso de CPU")

        # Redesenhar os gr ficos
        plt.pause(0.01)
        plt.legend()
        plt.draw()
```

2.5. Análise de anomalias x tempo

Além do gráfico em tempo real das variáveis utilizadas no treinamento do modelo de detecção de anomalias, foi plotado um gráfico de anomalias x tempo. Esse gráfico permitiu ao gerente visualizar as anomalias detectadas ao longo do tempo, facilitando a identificação de possíveis causas e a tomada de decisões corretivas.

O gráfico de anomalias x tempo é uma ferramenta importante para a análise dos resultados obtidos pelo modelo de detecção de anomalias. Com essa visualização, foi possível identificar padrões de anomalias ao longo do tempo, permitindo uma análise mais aprofundada dos dados e a identificação de possíveis causas. Dessa forma, o gerente pôde tomar medidas corretivas mais rapidamente, evitando prejuízos ou riscos.

2.5.1. Código Análise de anomalias x tempo

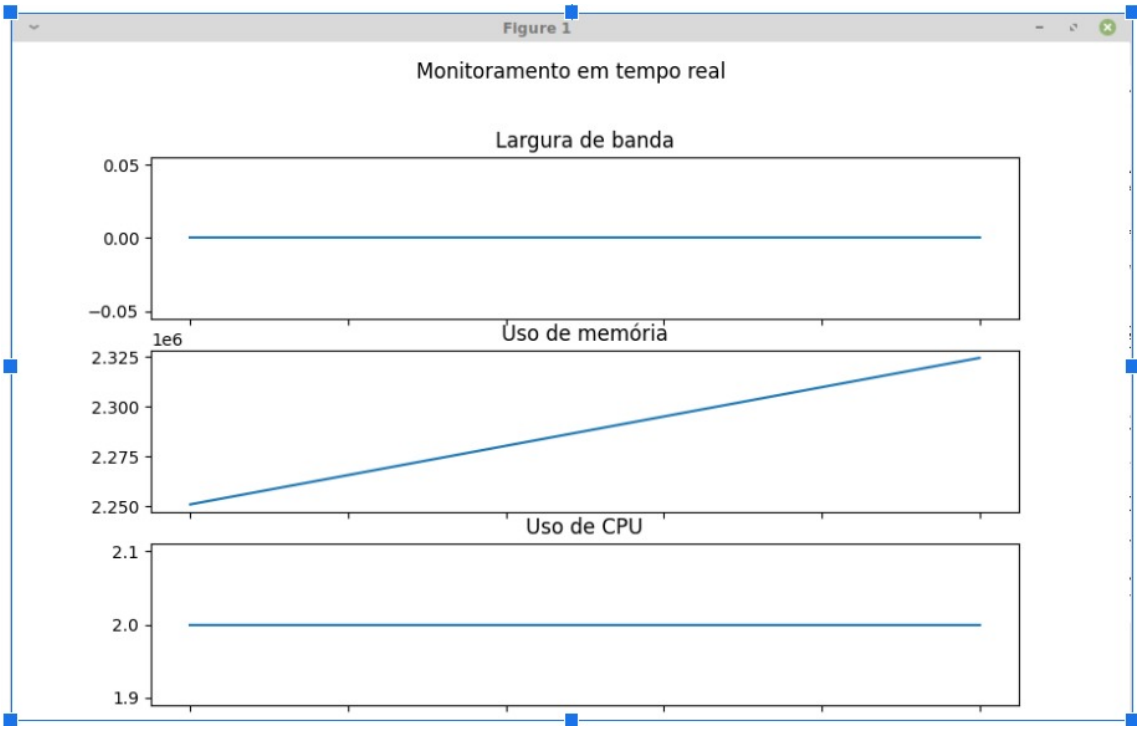

Figura 4. Gráfico Monitoramente em tempo real

```
    def anomaly_per_time(self,timestamps,predictions):
        import matplotlib.pyplot as plt

        anomaly_timestamps = [timestamp for timestamp, prediction in zi
        plt.scatter(anomaly_timestamps, [-1] * len(anomaly_timestamps),
        plt.xlabel('Tempo')
        plt.ylabel('Anomalia')
        plt.legend()
        plt.draw()
```

2.6. Funcionamento geral do programa

Ao final da execução do programa, é possível visualizar na tela as informações referentes às features mais importantes e ao gráfico em tempo real. Essa visualização permite uma análise mais detalhada dos dados e contribui para a tomada de decisões mais precisas pelo gerente.

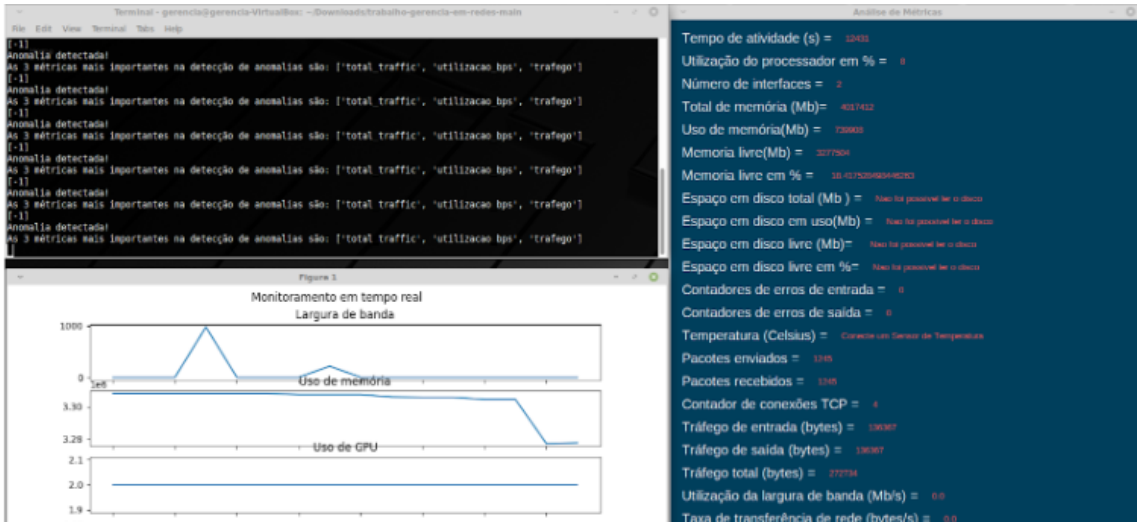

Figura 5. Print do funcionamento geral do programa

3. Referências

https://www.researchgate.net/figure/Isolation-Forest-learned-iForest-construction-for-toy-datasetfig1352017898

http://www.linux-admins.net/2012/02/linux-snmp-oids-for-cpumemory-and-disk.html

https://iphostmonitor.com/mib/tags/temperature-status.html

https://easysnmp.readthedocs.io/en/latest/

https://www.pysimplegui.org/