

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
PUC Minas Poços de Caldas
Ciência da computação

Samuel Tunes
Maria Amélia Doná Aguilar

Comunicação Pyhton x NodeJS

Poços de Caldas
2023

Lista de ilustrações

Figura 1 – Código em Python	4
Figura 2 – Terminal de execução do código em python	6
Figura 3 – Código em NodeJS	7
Figura 4 – Terminal de execução do código em NodeJS	8

1 Introdução

Foi apresentado na aula de Laboratório de Redes de Computadores, ministrada pelo professor Harison, onde a proposta foi realizar a comunicação de um script Python (utilizando Flask) com um script NodeJS da lista do dia 27 de abril. A seguir foi pedido que aprimore esses códigos para atingir tais objetivos

- Transmitir dados através de requisições REST;
- Transmitir dados como JSON;
- Realizar a comunicação em ambos sentidos (NodeJS para Python e Python para NodeJS);
- Desenvolver de forma que o NodeJS possa ser usado como uma interface simplificada para o usuário e o Python que deve realizar processamentos característicos de Python (exemplos-sugestões: o Python pode executar um código de teste de machine learning com alguma base disponível abertamente online; o Python pode ser usado para fazer web scraping; o Python pode ser usado para qualquer processamento longo);
- O NodeJS não pode ficar “parado esperando” o retorno do Python. Ou seja, o NodeJS deve liberar a GUI;

Com esses objetivos foi disponibilizado fazer o trabalho individualmente ou em dupla e que no final deveria ser entregue a documentação do funcionamento do código, tal é esta a documentação e um vídeo funcional do projeto.

2 Desenvolvimento

Os códigos que apresentados a seguir atende aos objetivos propostos. O código em Python utiliza o framework Flask para criar uma API REST que realiza o processamento de machine learning e fornece os resultados, como a acurácia e a média das idades. O código em Node.js faz requisições para a API Python e exibe os resultados no console.

A comunicação entre os dois scripts é realizada por meio de requisições HTTP utilizando JSON como formato de dados. O Node.js inicia o processamento no script Python, mas não fica esperando pelo retorno, permitindo que a interface do Node.js permaneça liberada.

2.1 Python

O código em Python a seguir é um exemplo de aplicação web utilizando o framework Flask. Onde terá a explicação breve de como funciona cada parte do código:

Figura 1 – Código em Python

```
server.js x client.py x
1 from flask import Flask, jsonify
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.metrics import accuracy_score
5
6 import threading
7 import time
8 import pandas as pd
9
10 #idade: Representa a idade do cliente em anos.
11 #premio: Representa o valor do prêmio de seguro pago pelo cliente.
12 #seguro: Indica se o cliente teve um sinistro de seguro (1) ou não (0)
13 data = [
14     [35, 1000, 1],
15     [42, 2500, 1],
16     [28, 1800, 0],
17     [50, 3200, 1],
18     [45, 2800, 1],
19 ]
20 columns = ['idade', 'premio', 'seguro']
21 df = pd.DataFrame(data, columns=columns)
22 X = df[['idade', 'premio']]
23 y = df['seguro']
24 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
25
26 app = Flask(__name__)
27
28 @app.route('/aluno', methods=['GET'])
29 def get_aluno():
30     aluno = {
31         'nome': 'Maria, Samuel',
32         'codigoMatricula': '1278625, 1303214'
33     }
34     return jsonify(aluno)
35
36 @app.route('/')
37 def home():
38     return 'Servidor Python em execução!'
39
40 def processar():
41     # Simula um processamento demorado
42     print('Iniciando processamento...')
43     time.sleep(5)
44     modelo = LogisticRegression()
45     modelo.fit(X_train, y_train)
46
47     previsoes = modelo.predict(X_test)
48
49     acuracia = accuracy_score(y_test, previsoes)
50     media = df['idade'].mean()
51     print('Acurácia:', acuracia)
52     print('Média de idade:', media)
53     print('Processamento concluído!')
54
55 @app.route('/processar', methods=['GET'])
56 def iniciar_processamento():
57
58     threading.Thread(target=processar).start()
59     return jsonify({'mensagem': 'Processamento iniciado.'})
60
61 if __name__ == '__main__':
62     app.run(port=8000)
```

Importação de bibliotecas: o código começa importando as bibliotecas necessárias (como o Flask para criar um aplicativo da Web) e outras bibliotecas de aprendizado de máquina, como o sklearn.

Definição de dados: Em seguida, definimos um conjunto de dados como uma lista

bidimensional. Cada linha representa uma instância de dados e cada coluna representa uma propriedade ou atributo dessas instâncias. Nesse caso, as informações incluem idade, prêmio de seguro e uma variável de destino que indica se o cliente recebeu compensação de seguro.

Crie um DataFrame: os dados são então convertidos pela biblioteca pandas em um objeto DataFrame. Um DataFrame é uma estrutura de dados tabulares que permite organizar e processar dados de forma conveniente.

Divisão de dados: Os dados são divididos em conjuntos de treinamento e teste usando a função `'train_test_split'` da biblioteca sklearn. Essa distribuição é comum em tarefas de aprendizado de máquina para avaliar a capacidade de um modelo de generalizar para novos dados.

Definição de aplicação Flask: Um aplicativo Flask é criado usando o construtor Flask e atribuído à variável `'app'`. Flask é um framework leve e flexível para construir aplicações web em Python.

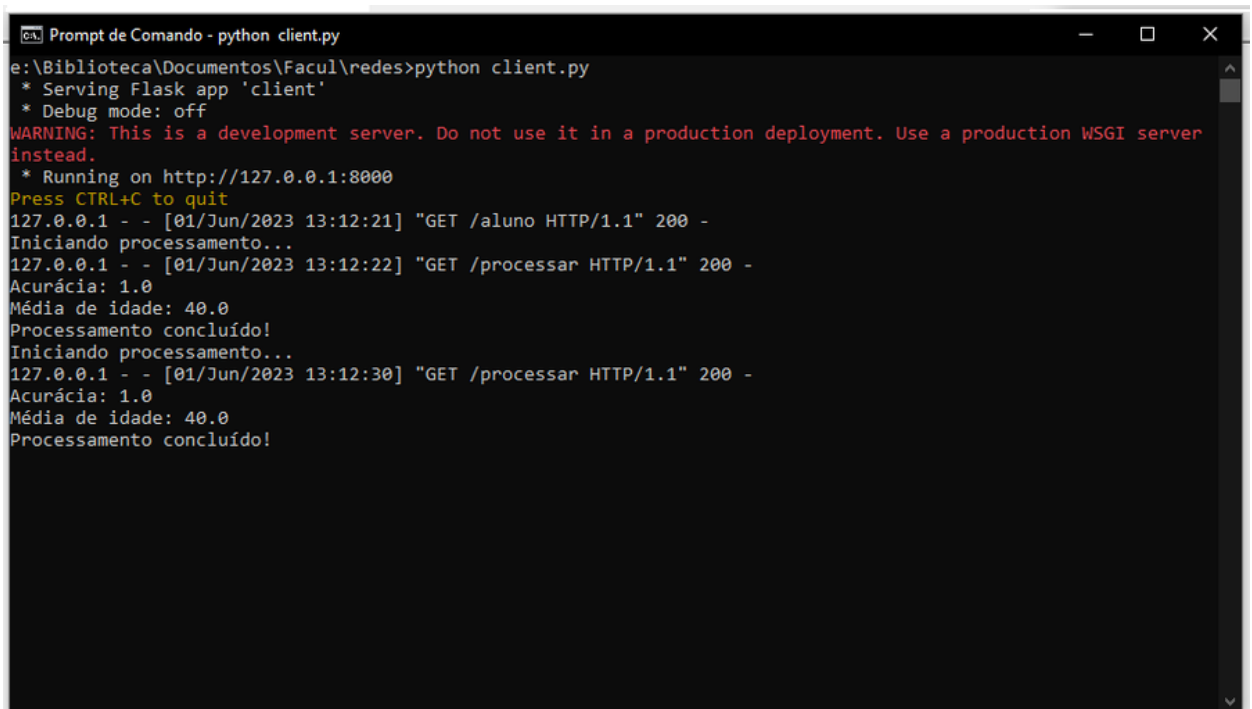
Definição de rota: as rotas são definidas para o aplicativo Flask. Cada rota está associada a uma ação que é executada quando a rota é usada. Por exemplo, a rota `'/aluno'` está vinculada à função `'get_aluno'`, que retorna informações sobre o aluno.

Definição da função de processamento: É definida uma função de processamento que simula um processamento longo (fornecido com um atraso de 5 segundos) e executa o treinamento do modelo de regressão logística usando os dados de treinamento.

Rota de início do processo: especifica a rota `'/processar'` que inicia a operação de processamento em um segmento específico. Isso permite o processamento em segundo plano enquanto a resposta está sendo enviada ao cliente.

Execução do servidor: Finalmente, o servidor Flask é iniciado chamando o método `'run'` da aplicação Flask e o servidor começa a ouvir as solicitações HTTP.

Resumindo, o código cria um aplicativo web Flask que abre vários caminhos para se comunicar com o servidor. Se a rota `/processor` for usada, o processo de treinamento do modelo de aprendizado de máquina será iniciado quando o servidor estiver disponível para outras solicitações, o resultado pode ser visto a baixo:

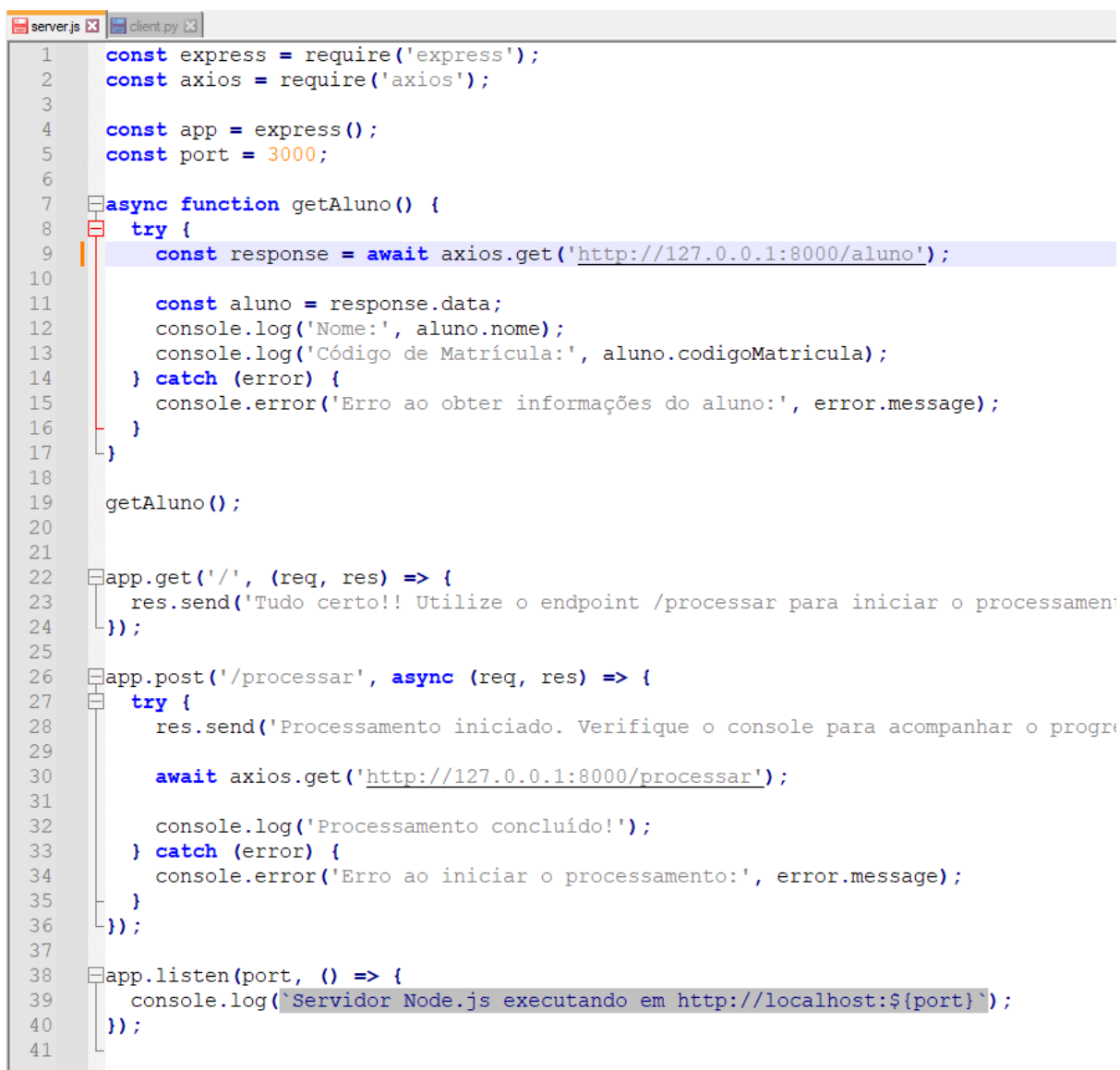
Figura 2 – Terminal de execução do código em python

```
Prompt de Comando - python client.py
e:\Biblioteca\Documentos\Facul\redes>python client.py
* Serving Flask app 'client'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
instead.
* Running on http://127.0.0.1:8000
Press CTRL+C to quit
127.0.0.1 - - [01/Jun/2023 13:12:21] "GET /aluno HTTP/1.1" 200 -
Iniciando processamento...
127.0.0.1 - - [01/Jun/2023 13:12:22] "GET /processar HTTP/1.1" 200 -
Acurácia: 1.0
Média de idade: 40.0
Processamento concluído!
Iniciando processamento...
127.0.0.1 - - [01/Jun/2023 13:12:30] "GET /processar HTTP/1.1" 200 -
Acurácia: 1.0
Média de idade: 40.0
Processamento concluído!
```

2.2 NodeJS

O código em Node.js a seguir é um exemplo de aplicação web utilizando o framework Express. Onde será explicado de forma breve como funciona cada parte do código:

Figura 3 – Código em NodeJS



```
1  const express = require('express');
2  const axios = require('axios');
3
4  const app = express();
5  const port = 3000;
6
7  async function getAluno() {
8    try {
9      const response = await axios.get('http://127.0.0.1:8000/aluno');
10
11      const aluno = response.data;
12      console.log('Nome:', aluno.nome);
13      console.log('Código de Matrícula:', aluno.codigoMatricula);
14    } catch (error) {
15      console.error('Erro ao obter informações do aluno:', error.message);
16    }
17  }
18
19  getAluno();
20
21
22  app.get('/', (req, res) => {
23    res.send('Tudo certo!! Utilize o endpoint /processar para iniciar o processamento');
24  });
25
26  app.post('/processar', async (req, res) => {
27    try {
28      res.send('Processamento iniciado. Verifique o console para acompanhar o progresso');
29
30      await axios.get('http://127.0.0.1:8000/processar');
31
32      console.log('Processamento concluído!');
33    } catch (error) {
34      console.error('Erro ao iniciar o processamento:', error.message);
35    }
36  });
37
38  app.listen(port, () => {
39    console.log(`Servidor Node.js executando em http://localhost:${port}`);
40  });
41
```

Importação de bibliotecas: o código começa importando as bibliotecas necessárias, por exemplo, o Express para criar a aplicação web e o Axios para fazer requisições HTTP.

Criando um aplicativo Express: Um aplicativo Express é criado chamando a função 'express()' e atribuindo o objeto resultante à variável do 'app'. Express é um framework web rápido e minimalista para Node.js.

Definindo rotas: Existem duas rotas definidas para um aplicação Express. A rota '/' responde às solicitações GET dizendo "OK! Inicie o processamento usando o terminal /processar". A rota '/processar' responde a solicitações POST.

Função 'GetAluno': Definimos uma função 'getAluno' que utiliza o Axios para fazer uma requisição GET para a rota '/aluno' em outro servidor (servidor Python). As informações retornadas pela rota são exibidas no console.

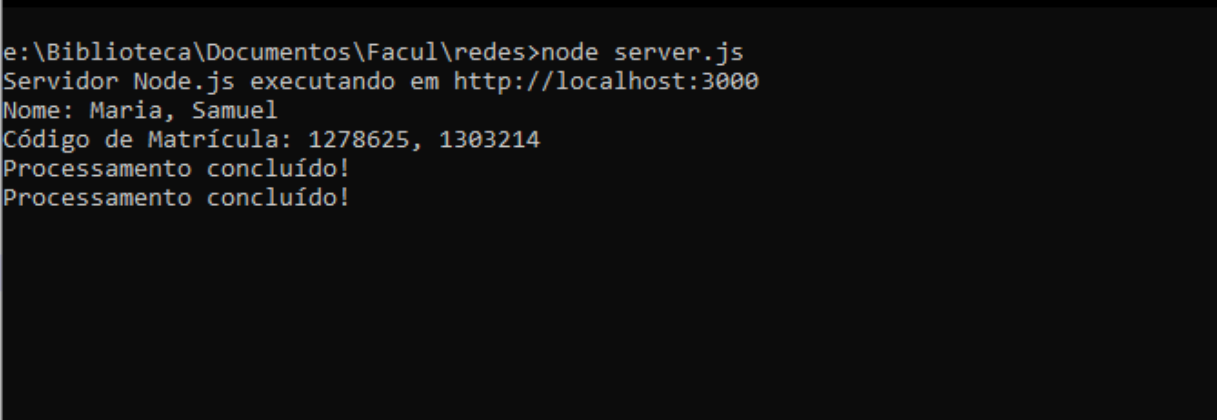
Rota '/processar': A rota '/processar' responde a requisições POST. O uso dessa rota

enviará imediatamente uma resposta ao cliente com a mensagem “O processamento foi iniciado. Verifique o andamento do console.” Em seguida, uma solicitação GET é feita para o caminho do servidor Python `/processar` usando Axios. A resposta do servidor Python é exibida no console.

Iniciando o servidor: O servidor Express é iniciado chamando o método `'listen'` e especificando a porta na qual o servidor deve atender as solicitações.

Resumidamente, o código cria um aplicativo da Web Express com duas rotas: a rota `'/'` exibe a mensagem de boas-vindas e a rota `/processar` inicia o processamento quando uma solicitação é enviada ao servidor Python. Além disso, o código inclui uma função para fazer uma solicitação GET ao servidor Python e exibir os dados retornados, a seguir terá o resultado do terminal.

Figura 4 – Terminal de execução do código em NodeJS

A screenshot of a terminal window with a black background and white text. The text shows the execution of a Node.js script. The first line is a command prompt: `e:\Biblioteca\Documentos\Facul\redes>node server.js`. The subsequent lines are the output of the script: `Servidor Node.js executando em http://localhost:3000`, `Nome: Maria, Samuel`, `Código de Matrícula: 1278625, 1303214`, and two lines of `Processamento concluído!`.

```
e:\Biblioteca\Documentos\Facul\redes>node server.js
Servidor Node.js executando em http://localhost:3000
Nome: Maria, Samuel
Código de Matrícula: 1278625, 1303214
Processamento concluído!
Processamento concluído!
```

3 Conclusão

Para finalizar, acredito que concluímos corretamente todos os objetivos propostos para este trabalho. Com que fizemos a comunicação Python x NodeJS e todos os outros inclementos pedidos.

A comunicação funciona de forma correta, o 'client', Python, espera ser acessado ou chamado pelo servidor e garante diversar funções, como o retorno da 'struct' aluno em que nela está os nomes dos integrantes do grupo e o código de pessoa para o servidor. Em seguida espera uma chamada para executar o método '/processar' em que nele irá calcular a acurácia e a idade média de um data frame previamente criado.

No caso do Node, ele é o servidor que conectará com o python e irá esperar uma chamada, por exemplo pelo código no terminal "curl -X POST http://localhost:3000/processar", que aciona a função do 'server' e do servidor irá executar o 'client', Python. Por fim retornará uma mensagem de 'Processamento concluído!' ou no meio da execução dirá que não foi possível.