Bootcamp IGTI: Desenvolvedor Python

Desafio Final

Módulo 5	**Desafio Final**

Objetivos

Exercitar os seguintes conceitos trabalhados no Módulo:

- ✓ Scikit-Learning.
- ✓ Pandas/Numpy.
- ✓ Flask.

Enunciado

Neste desafio final são utilizados todos os módulos apresentados durante o bootcamp de desenvolvedor Python. Módulos como o scikit-learn, pandas, numpy e flask são empregados para realizar o “deploy” de um modelo de *machine learning*. O *deploy* de um modelo consiste em colocá-lo “em produção”. Desse modo é possível que diferentes usuários possam interagir com a aplicação através do envio de dados e recebimento de informação.

Para a solução deste desafio, você deve construir e treinar um modelo de aprendizado de máquina que consiga prever se um determinado indivíduo desenvolverá ou não diabetes. Após os ajustes necessários, você deve construir uma interface, utilizando o *flask*, que possibilite ao usuário introduzir os dados necessários para que o modelo possa indicar se as características fornecidas sugerem ou não o desenvolvimento de um quadro de diabetes.

Atividades

Os alunos deverão desempenhar as seguintes atividades:

1. Acessar a IDE de desenvolvimento desejada.
2. Baixar o dataset presente no link:

   https://drive.google.com/drive/folders/13ezZUhmZKm_4lnfnt5dKeTzEn6ZEHa7I?usp=sharing.

3. Responder às questões presentes neste desafio.

Observaçoes:

O dataset utilizado possui as seguintes colunas:

0. Número de vezes em que ficou grávida.
1. Concentração de glicose.
2. Pressão diastólica (mm Hg).
3. Espessura da dobra cutânea do tríceps (mm).
4. Insulina (mu U/ml).
5. Índice de massa corporal (peso em kg/(altura em m)^2).
6. Histórico familiar de diabetes.
7. Idade (anos).
8. Classificação (0 ou 1 - 0 não diabético / 1 diabético).

Este dataset não possui um cabeçalho definido. Desse modo, você deve utilizar a leitura, através do módulo pandas, fornecendo o parâmetro que **NÃO** considera a primeira linha como cabeçalho.

Para as perguntas referentes aos modelos, utilize:

Algoritmo KNN:

```
clf_KNN = KNeighborsClassifier(n_neighbors=5)
```

Algoritmo Árvore de Decisão

```
clf_arvore = DecisionTreeClassifier(random_state=1)
```

Algoritmo Rede MLP

```
clf_mlp = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5,
10), random_state=1)
```

- Para a aplicação dos algoritmos, utilize como entrada as colunas: ***0, 1, 2, 3, 4, 5, 6 e 7***. A saída para os algoritmos deve ser a coluna ***8***.
- Utilize, para normalização dos dados, as definições:

```
normaliza = MinMaxScaler() #objeto para a normalização
entradas_normalizadas=normaliza.fit_transform(entradas)
```

- Utilize, para divisão entre treinamento e teste do algoritmo, as definições:

  ```
  train_test_split(entradas_normalizadas, saida,
  test_size=0.30,random_state=42)
  ```

- Utilize esta sequência de operações para chegar no resultado final: divida os dados entre entrada e saída, normalize apenas as entradas utilizando o ***MinMaxScaler*** e, depois, aplique a divisão entre treinamento e teste com o ***train_test_split***.
- Utilize os dados de “teste” para avaliar as previsões de classificação dos modelos.

Após selecionar o melhor modelo (maior acurácia) dentre os apresentados acima, utilize o código abaixo para “salvar” este modelo. Após salvar o modelo, ele pode ser utilizado para prever a saída após novas entradas. O módulo a ser utilizado é o “joblib”. A Figura 1 apresenta como esse módulo pode ser utilizado.

Figura 1 – Utilização do módulo “joblib”.

```
import joblib

#salvando o modelo no disco
nome_do_arquivo = 'melhor_modelo.sav'  #observem a extensão ".sav"
joblib.dump(melhor_modelo, nome_do_arquivo)  # melhor_modelo = modelo com maior acurácia
                                             # nome_do_arquivo = caminho do local onde deve ser salvo o modelo


#carregando o modelo
modelo_salvo = joblib.load(nome_do_arquivo)  #realiza a carga do modelo salvo
```

Para a construção da interface gráfica, utilize como base o código html presente na Figura 2:

Figura 2 – Código base html a ser utilizado para construção da interface.

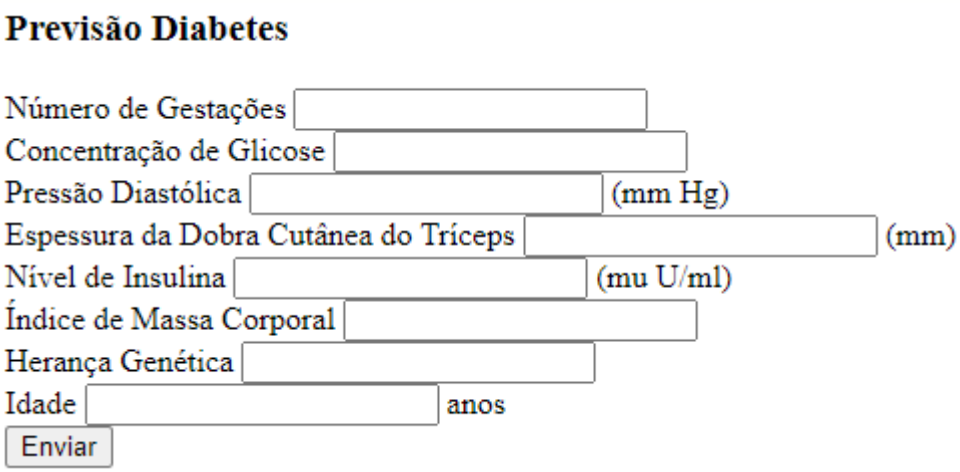
```
<html>
<body>
    <h3>Previsão Diabetes</h3>

<div>
<form action="/result" method="POST">
    <label for="gest">Número de Gestações</label>
    <input type="text" id="gest" name="gest">
    <br>
    <label for="glic">Concentração de Glicose</label>
    <input type="text" id="glic" name="glic">
    <br>
    <label for="age">Idade</label>
    <input type="text" id="age" name="age"> anos
    <br>
    <input type="submit" value="Enviar">
</form>
</div>
</body>
</html>
```

Como pode ser visto, nem todo o código foi mostrado. O complemento das demais linhas é parte deste desafio. Você pode alterar o código para deixá-lo mais amigável ao usuário, entretanto, as questões referentes ao código devem ser respondidas utilizando esse esboço sem modificações.

A Figura 3 apresenta um exemplo de exibição da saída html.

Figura 3 – Modelo de formulário a ser enviado.



Crie uma função em python que receba os dados enviados pelo formulário presente na Figura 3. A Figura 4 apresenta esta função.

Figura 4 – Função utilizada para obter os dados digitados pelo usuário.

```
1  import numpy as np
2  import joblib
3  from flask import Flask, request, jsonify, render_template
4
5  app = Flask(__name__)
6
7
8  def previsao_diabetes(lista_valores_formulario):
9      prever=[redacted].reshape(1,8)      #transforma os valores do formulario
10     modelo_salvo = joblib.load('melhor_modelo.sav')      #realiza a carga do modelo salvo
11     resultado =[redacted].predict(prever)      #aplica a previsao
12     return resultado[0]
```

Complemente o código presente na Figura 4 com o código presente na Figura 5. Esse código corresponde à construção de uma aplicação web utilizando o flask. Por meio dessa aplicação é possível receber os dados do formulário e utilizá-los no modelo construído.

Figura 5 – Códigos para a construção da aplicação web com Flask.

```
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/result',methods=['POST'])
def result():
    if request.method=='POST':
        lista_formulario=request.form.to_dict()
        lista_formulario=list(lista_formulario.values())
        lista_formulario= list(map(                    ))
        resultado=previsao_diabetes(lista_formulario)
        if int(resultado)==1:
            previsao='Possui diabetes'
        else:
            previsao='Nao possui diabetes'

        #retorna o resultado para uma pagina html
        return render_template("resultado.html",previsao=previsao)

if __name__ == "__main__":
    app.run(debug=True)
```

Para a execução de todo o programa, também é necessário desenvolver um arquivo HTLM (“resultado.html”) que receba a previsão realizada pelo algoritmo.