Trabalho de Machine Learning

Segundo Trabalho Aprendizado de Máquina e Modelagem de Conhecimento Incerto

Alunos: João Pedro Alkamim
Sarah Anduca de Oliveira

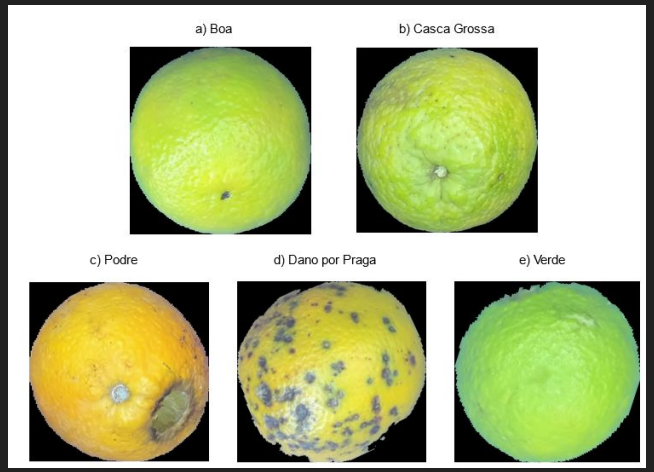
O Problema

A produção de laranjas e seus derivados, como suco, polpa, óleos essenciais e essências, compõem um mercado lucrativo que representa uma parte significativa da economia de diversos países. O setor citrícola possui um PIB de US$ 6,5 bilhões (em 2009), sendo US$ 4,3 bilhões no mercado interno e US$ 2,1 bilhões no mercado externo. Essa indústria gera milhares de empregos diretos e indiretos, totalizando cerca de 230 mil posições com um salário total de R$ 676 milhões.

O objetivo deste projeto é desenvolver um sistema inovador que utilize uma combinação de técnicas de PDI e AM para predizer com precisão as classes de laranjas em um conjunto de dados. O sistema é capaz de extrair características de textura de imagens usando a técnica LPQ e alimentar essas informações para os classificadores gerados pelos algoritmos K-NN. O trabalho prático irá comparar esse resultado com as técnicas SVM e Floresta Randômica. As classes de laranjas que o sistema foi projetado para predizer incluem boa, casca-grossa, dano por praga, podre e verde.

Banco de Dados

Para o desenvolvimento do classificador, é necessário criar um conjunto de dados composto por imagens de laranjas que serão utilizadas nas fases de treinamento e teste do modelo de aprendizagem de máquina. São cinco as categorias: boa, com casca grossa, danificada por pragas, podre e verde. Abaixo é apresentado um exemplo de cada uma dessas classes.



Leitura dos dados e extração das características

Para a extração das características foi usada a técnica LPQ (Local Phase Quantization) com a linguagem MATLAB. O resultado desta extração são vários arquivos .txt com as características das texturas das laranjas.

Foi feito então um código em python para colocar todos esses dados e suas respectivas classes em um banco de dados. Foram usadas 11.060 laranjas, sendo 2.212 para cada tipo.

Leitura dos dados e extração das características

O Local Phase Quantization (LPQ) é uma técnica de processamento de imagem que é usada para extrair recursos de textura de uma imagem. A ideia básica é calcular a fase local da imagem e usar essa informação para codificar a textura da imagem em um histograma de 256 valores.

O parâmetro winSize controla o tamanho da janela local usada para calcular a fase da imagem. Ele deve ser um número ímpar e igual ou superior a 3. Valores maiores de winSize resultam em uma descrição de textura mais suave e, portanto, menos discriminatória, enquanto valores menores podem ser muito sensíveis a ruídos.

Os valores 3, 5, 7 e 9 para winSize são usados para testar diferentes tamanhos de janela local e avaliar a sensibilidade do LPQ a diferentes níveis de ruído. Valores menores de winSize são mais adequados para imagens com texturas finas e detalhadas, enquanto valores maiores são mais adequados para imagens com texturas mais grossas e uniformes.

Leitura dos dados e extração das características

```
function LPQdesc = lpqNEW(img,winSize,decorr,freqestim,mode)
% Funtion LPQdesc=lpq(img,winSize,decorr,freqestim,mode) computes the Local Phase Quantization (LPQ) descr
% for the input image img. Descriptors are calculated using only valid pixels i.e. size(img)-(winSize-1).
%
% Inputs: (All empty or undefined inputs will be set to default values)
% img = N*N uint8 or double, format gray scale image to be analyzed.
% winSize = 1*1 double, size of the local window. winSize must be odd number and greater or equal to 3 (de
% decorr = 1*1 double, indicates whether decorrelation is used or not. Possible values are:
%                     0 -> no decorrelation,
%           (default) 1 -> decorrelation
% freqestim = 1*1 double, indicates which method is used for local frequency estimation. Possible values a
%               (default) 1 -> STFT with uniform window (corresponds to basic version of LPQ)
%                         2 -> STFT with Gaussian window (equals also to Gaussian quadrature filter pair)
%                         3 -> Gaussian derivative quadrature filter pair.
% mode = 1*n char, defines the desired output type. Possible choices are:
%        (default) 'nh' -> normalized histogram of LPQ codewords (1*256 double vector, for which sum(resul
%                  'h'  -> un-normalized histogram of LPQ codewords (1*256 double vector)
%                  'im' -> LPQ codeword image ([size(img,1)-r,size(img,2)-r] double matrix)
%
% Output:
% LPQdesc = 1*256 double or size(img)-(winSize-1) uint8, LPQ descriptors histogram or LPQ code image (see
%
% Example usage:
% img=imread('cameraman.tif');
% LPQhist = lpq(img,3);
% figure; bar(LPQhist);
%

% Version published in 2010 by Janne Heikkil, Esa Rahtu, and Ville Ojansivu
% Machine Vision Group, University of Oulu, Finland

```

Leitura dos dados e extração das características

Para utilizar os resultados do LPQ em um classificador, foi necessário armazená-los em um banco de dados. Cada conjunto de resultados foi separado por atributo e classe correspondente da data (dados) da laranja, e então armazenado em uma pasta. As matrizes de dados X e rótulos y foram inicializadas e percorridas para cada classe. Em seguida, os arquivos dentro de cada pasta foram lidos e os dados foram adicionados à matriz X, enquanto a classe correspondente foi adicionada à matriz y. Por fim, os dados foram salvos em arquivos.

O número de atributos em cada arquivo foi definido como 256, e o número de classes foi definido como cinco, incluindo "Boa", "CascaGrossa", "Podre", "Praga" e "Verde".

Leitura dos dados e extração das características

```python
# Defina as classes
classes = ['Boa', 'CascaGrossa', 'Podre', 'Praga', 'Verde']

# Defina o número de atributos em cada arquivo
num_atributos = 256

# Inicialize as matrizes de dados X e os rótulos y
X = np.zeros((0, num_atributos))
y = np.zeros((0,))

# Percorra cada classe e leia os arquivos
for classe_idx, classe in enumerate(classes):
    print(classe)
    # Obtenha uma lista de todos os arquivos na pasta
    files = os.listdir(f'variosFold/Resultado9/{classe}')
    # Percorra cada arquivo na pasta
    for file in files:
        # Leia o arquivo
        data = np.loadtxt(f'variosFold/Resultado9/{classe}/{file}')
        # Adicione os dados à matriz X e o rótulo da classe à matriz y
        X = np.vstack((X, data))
        y = np.append(y, classe_idx)

print(f'Tamanho da matriz X: {X.shape}')
print(f'Tamanho da matriz y: {y.shape}')

# Salve os dados em arquivos
np.save('Banco/X9.npy', X)
np.save('Banco/y9.npy', y)
```

Classificadores

Neste estudo serão empregados três classificadores distintos: KNN, SVM e Floresta Randômica (Random Forest). A comparação entre esses modelos será feita com base nos resultados obtidos a partir do KNN, que já apresentou o melhor atributo para o LPQ.

Por essa razão, utilizaremos apenas esse atributo em todos os outros classificadores, juntamente com os resultados do melhor valor de K encontrado.

Para avaliar o desempenho dos modelos, utilizaremos a técnica de validação cruzada junto com *holdout*, combinada com a função gridsearchCV, que realiza a busca em grade dos melhores parâmetros para cada classificador. Ao final do processo, serão identificados o melhor atributo para cada modelo, juntamente com a acurácia e o f1-score correspondentes.

Validação Cruzada

O método *Holdout* é utilizado para dividir um conjunto de dados em treinamento e teste, permitindo a criação e avaliação de desempenho de um modelo. Geralmente, é utilizado uma porcentagem de dados para treinamento e outra para teste, como 70% e 30%, respectivamente. Porém, em alguns casos, a quantidade de dados pode não ser suficiente para obter um modelo de boa qualidade.

Em vez disso, a validação cruzada é uma técnica mais eficiente, que divide o conjunto de dados em k subconjuntos, permitindo que o modelo seja treinado k vezes, com cada subconjunto sendo usado para teste em uma iteração e os demais subconjuntos para treinamento. Essa técnica produz k resultados de avaliação que podem ser combinados para estimar o desempenho do modelo, tornando-o mais robusto e confiável.

Validação Cruzada

Técnica *Holdout*

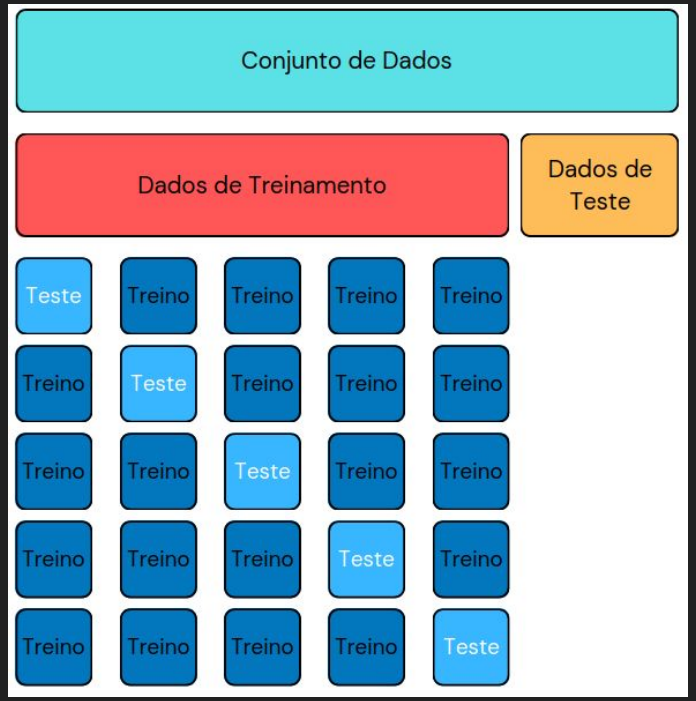
Técnica: Validação Cruzada

Validação Cruzada

É importante ressaltar que, ao utilizar o método holdout para separar os dados em treinamento e teste, pode ocorrer vazamento de informações, pois os dados de treinamento são utilizados também para avaliar o modelo. Para evitar esse problema, é comum utilizar uma combinação entre *holdout* e validação cruzada.

Na abordagem, os dados são divididos em treinamento e teste, e é realizada a validação cruzada nos dados de treinamento para encontrar o conjunto de hiperparâmetros que apresente a melhor acurácia. O teste final é feito nos 30% dos dados separados no início, usando o melhor conjunto de hiperparâmetros encontrado na validação cruzada.

Validação Cruzada



Técnica *Holdout* + Validação Cruzada

Resultados do KNN com o GridSearchCV

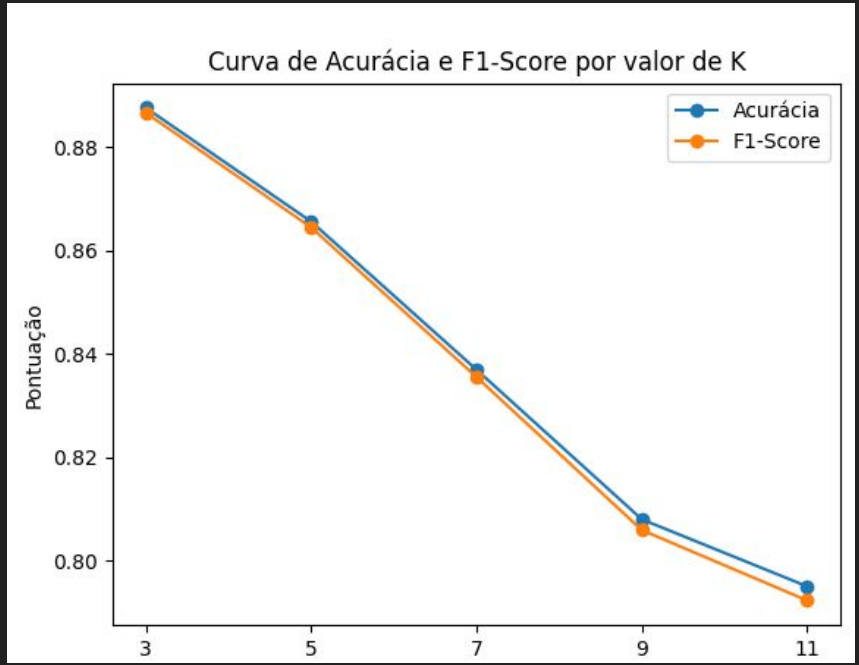
Melhor atributo do LPQ = 9

Resultados do K

Variáveis	K = 3	K = 5	K = 7	K = 9	K = 11
Acurácia	0,8876	0,8656	0,8369	0,8080	0,7951
Precisão	0,8883	0,8661	0,8369	0,8071	0,7934
Revocação	0,8871	0,8650	0,8363	0,8072	0,7941
F1-Score	0,8866	0,8644	0,8355	0,8059	0,7923

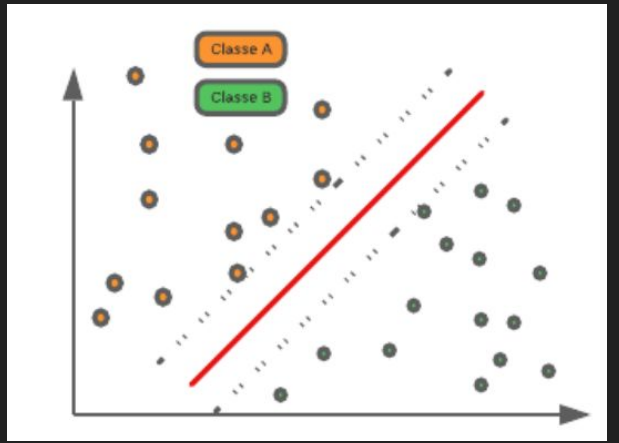
Resultados do KNN

Gráfico comparando os resultados dos K com acurácia e f1-score.



Resultados do SVM com o GridSearchCV

A máquina de vetores de suporte, ou Support Vector Machine (SVM), é um algoritmo de aprendizagem supervisionada que busca uma forma de agrupar os dados através de hiperplanos utilizando uma função kernel, visando predizer o rótulo da classe de uma nova instância. A Figura ilustra a divisão dos pontos laranjas e verdes que representam, respectivamente, instâncias da classe A e B, por uma linha vermelha que representa o hiperplano.



Resultados do SVM com o GridSearchCV

Melhor atributo do LPQ = 9

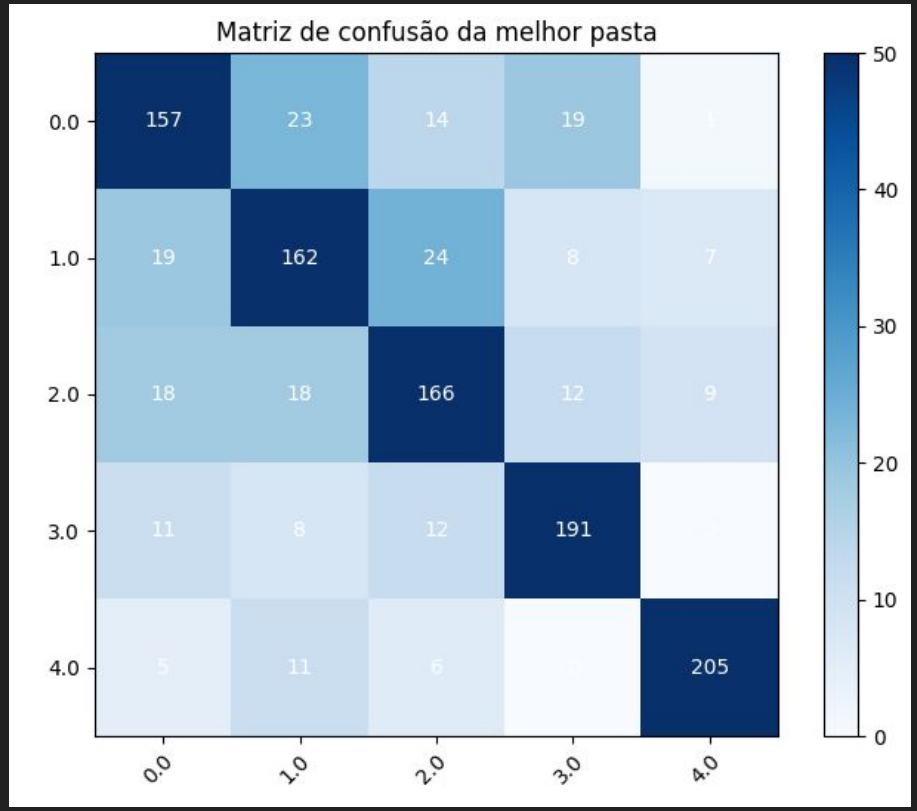
Atributos:

{ **Kernel: [‘rbf’], Gamma: [1, 0.1, 0.01, 0.001, 0.0001], C: [0.1, 1, 10, 100, 1000]}**

{ Kernel: [‘linear’], C: [1, 10, 100, 1000]}

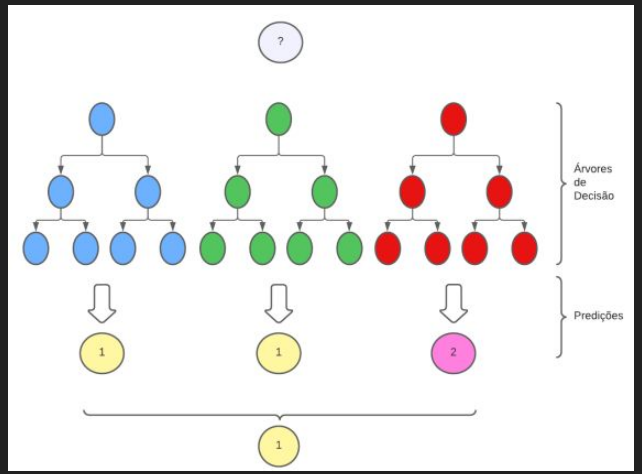
Variáveis	**C:** 1000, **Gamma:** 1, **Kernel:** rbf
Acurácia	0.76793
F1 - Score	0.76655

Resultados do SVM com o GridSearchCV



Resultados do RF com o GridSearchCV

A floresta randômica, ou Random Forest (RF), é um algoritmo de aprendizagem supervisionada que utiliza N árvores de decisão de forma aleatória de modo a realizar a classificação de novos objetos. Cada árvore contém regras que orientam as tomadas de decisão. A Figura mostra a classificação de uma nova instância pelo algoritmo utilizando três árvores de decisão para realizar a predição.



Resultados do Random Forest com o GridSearchCV

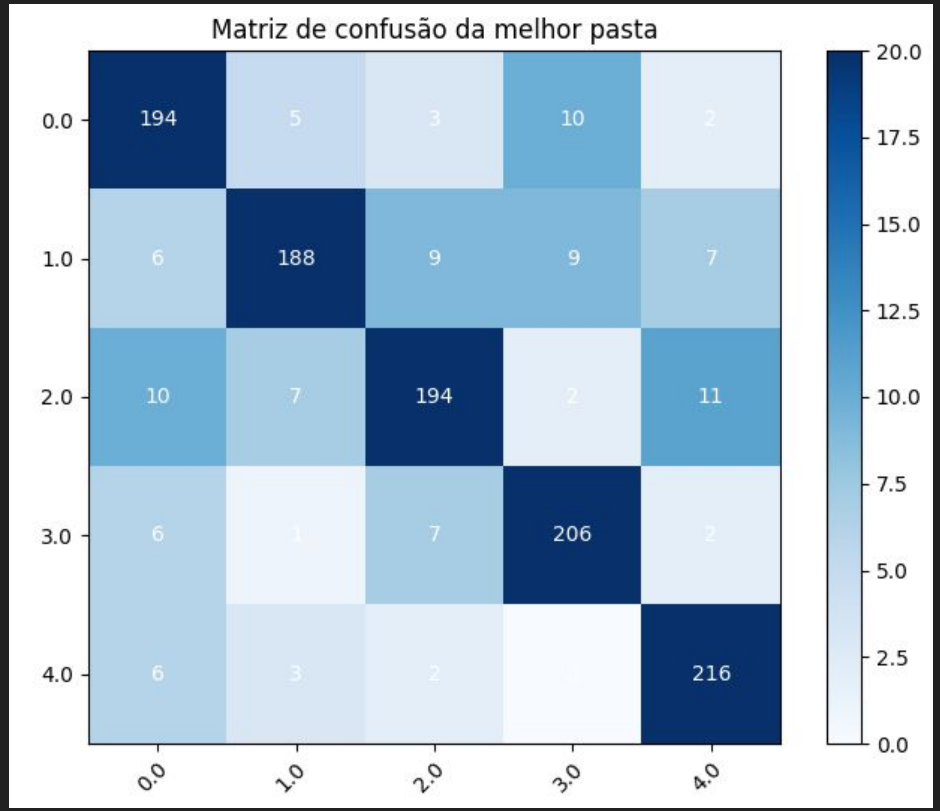
Melhor atributo do LPQ = 9

Atributos:

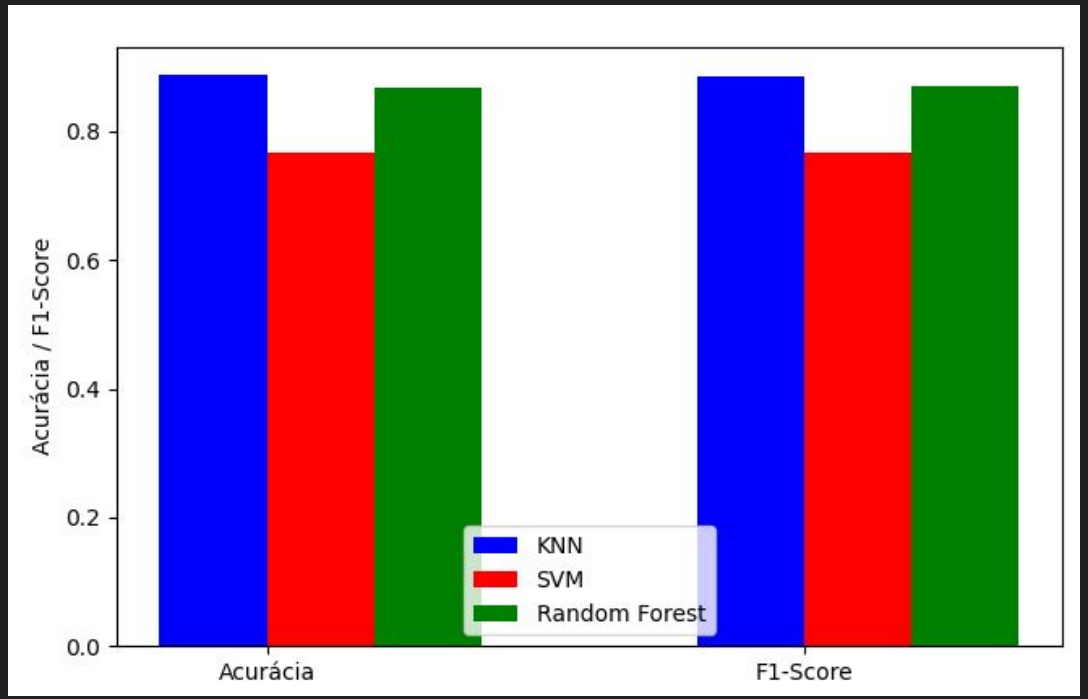
{ max_depth: [10, 50], min_samples_leaf: [1, 3], min_samples_split: [2, 5], n_estimators: [100, 200] }

Variáveis	**max_depth: 50, min_samples_leaf: 1, min_samples_split: 2, n_estimators: 200**
Acurácia	0.86842
F1 - Score	0.86950

Resultados do RF com o GridSearchCV



Comparação entre os classificadores



Analise

Para o KNN, a acurácia e o F1-score são bem próximos, indicando que o modelo tem um bom desempenho tanto na identificação correta das classes como na métrica balanceada. Além disso, as métricas têm valores altos, indicando que o modelo é capaz de fazer previsões com alta precisão.

Para o SVM, a acurácia e o F1-score são menores do que para o KNN, indicando que o modelo tem um desempenho pior na identificação correta das classes e na métrica balanceada. A diferença entre as duas métricas é pequena, o que sugere que o modelo não tem um problema de desbalanceamento de classes.

Para o Random Forest, a acurácia e o F1-score são bem próximos, assim como para o KNN, indicando que o modelo tem um bom desempenho tanto na identificação correta das classes como na métrica balanceada. Além disso, as métricas têm valores altos, indicando que o modelo é capaz de fazer previsões com alta precisão.

Referências

https://www.hermespardini.com.br/blog/?p=76

https://www.cdc.gov/mmwr/volumes/71/wr/mm7123a4.htm

https://www.textbookofcardiology.org/wiki/Chest_Pain_/_Angina_Pectoris

https://github.com/JPAlkamim/model_classificator