Lista 3 Aprendizado de Máquina

Link github: https://github.com/AHChaves/Aprendizado-de-Maquina

Q1 e Q2-

Utilizando dos códigos disponibilizados, primeiro carregamos a base restaurantev2.csv no script de tratamento da base, nele nós observamos os valores iniciais, excluímos colunas desnecessárias (ExÇ coluna Exemplo).

```
base = pd.read_csv('restaurantev2.csv', sep=';', encoding='latin-1')
```
✓ 0.0s

Aplicamos o LabelEncoder nas demais colunas, com exceção das colunas Tipo e conc.

```
cols_label_encode = ['Alternativo', 'Bar', 'Sex/Sab','fome', 'Cliente','Preço', 'Chuva', 'Res','Tempo']
base[cols_label_encode] = base[cols_label_encode].apply(LabelEncoder().fit_transform)
```
✓ 0.0s

Em seguida é aplicado o OneHotEncoder na coluna Tipo.

```
cols_onehot_encode = ['Tipo']
# Inicializar o OneHotEncoder (sparse_output=False retorna um array denso)
onehot = OneHotEncoder(sparse_output=False)

# Aplicar o OneHotEncoder apenas nas colunas categóricas
df_onehot = onehot.fit_transform(base[cols_onehot_encode])

# Obter os novos nomes das colunas após a codificação
nomes_das_colunas = onehot.get_feature_names_out(cols_onehot_encode)

# Criar um DataFrame com os dados codificados e as novas colunas
df_onehot = pd.DataFrame(df_onehot, columns=nomes_das_colunas)

# Combinar as colunas codificadas com as colunas que não foram transformadas
base_encoded= pd.concat([df_onehot, base.drop(columns=cols_onehot_encode)], axis=1)
```
✓ 0.0s

E por último, foi separado os valores utilizados para teste e treino, sendo um total de 20% da base para teste e 80% para treino.

```
X_treino, X_teste, y_treino, y_teste = train_test_split(X_prev, y_classe, test_size = 0.20, random_state = 42)
```

A implementação da árvore de decisão ocorreu em um outro arquivo, nele primeiro criamos o modelo utilizando o critério de gini, e posteriormente o critério de entropia como solicitado na questão 2.

```
modelo = DecisionTreeClassifier(criterion='gini')
modelo.fit(X_treino, y_treino)
```

realizamos a previsões de resultados:

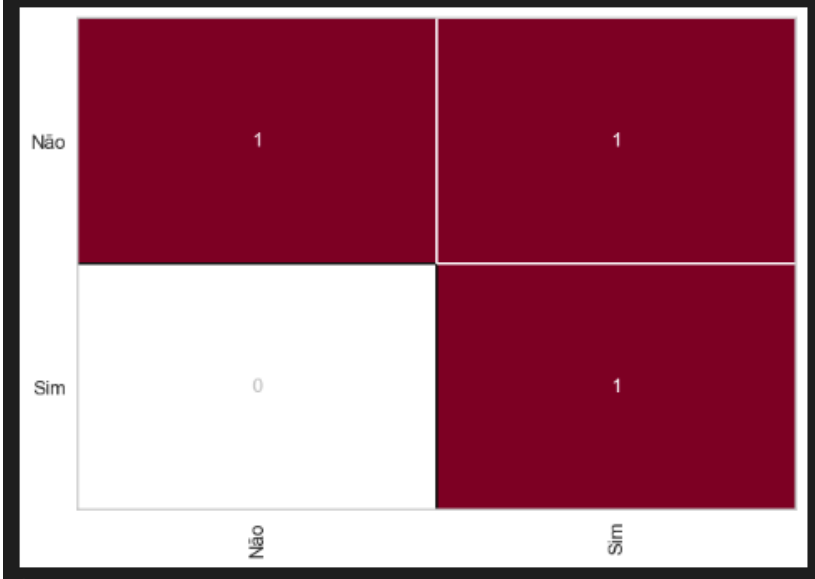
```
previsoes = modelo.predict(X_teste)
```

e utilizamos as funções de métrica do sklearn e a matriz de confusão do yellowbrick para analisar os resultados.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report # type: ignore
accuracy_score(y_teste,previsoes)
```

```
cm = ConfusionMatrix(modelo)
cm.fit(X_treino, y_treino)
cm.score(X_teste, y_teste)
```

```
/usr/lib/python3/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(

0.6666666666666666
```



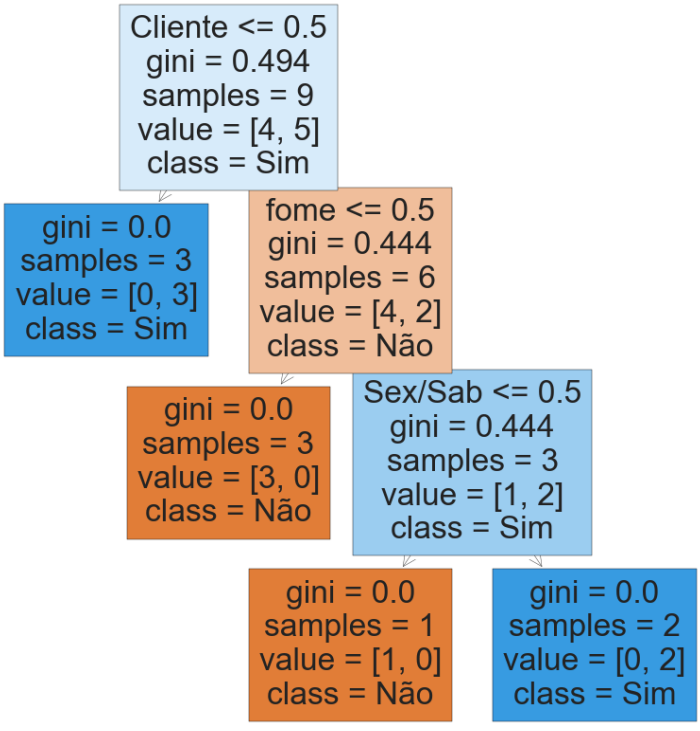
```
print(classification_report(y_teste, previsoes))
```

```
              precision    recall  f1-score   support

         Não       1.00      0.50      0.67         2
         Sim       0.50      1.00      0.67         1

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3
```

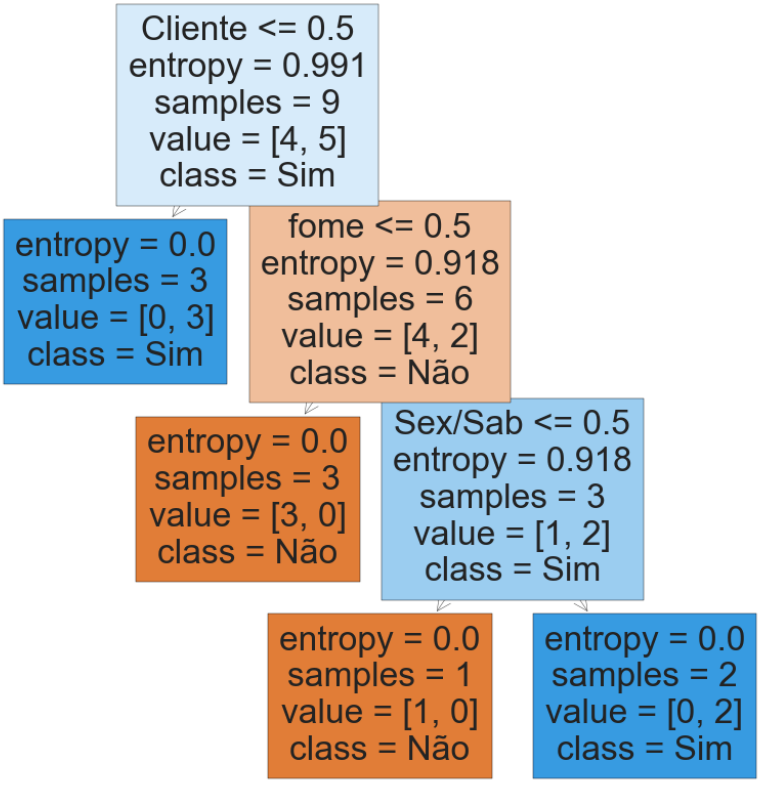
e por último plotamos a árvore para uma melhor visualização

```
from sklearn import tree
previsores = X_treino.columns
figura, eixos = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
tree.plot_tree(modelo, feature_names=previsores, class_names = modelo.classes_, filled=True);
```

gini:



entropia



O índice de Gini mede a probabilidade de um elemento ser classificado incorretamente se for escolhido aleatoriamente.

O cálculo de gini se dá pela fórmula

$$gini = 1 - (p1^2 + p2^2 \ldots pn^2)$$

sendo assim, as contas para determinar o coeficiente de gini da arvore foram:

raiz = $1 - (4/9^2 + 5/9^2) = 1 - 0{,}506 = 0{,}494$
nível 1 = $1 - (4/6^2 + 2/6^2) = 1 - 0{,}555\ldots = 0{,}444\ldots$
nível 2 = $1 - 1 - (1/3^2 + 2/3^2) = 1 - 0{,}555\ldots = 0{,}444\ldots$

Ambas as árvores, como podem ser observadas, são idênticas quanto ao resultado. A diferença das duas são os métodos utilizados para chegar em cada uma.

Q3 -

max_depth controla a profundidade máxima da árvore. Caso esse valor seja baixo a árvore pode não capturar padrões complexos. Se for muito grande, a árvore pode se tornar complexa demais, criando um overfitting dos dados.

max_features define quantas features(colunas) são consideradas. Se for muito pequeno a árvore pode ignorar características importantes. Se for muito grande, pode aumentar o risco de overfitting.

min_samples_leaf determina o número mínimo de amostras necessárias para que um nó seja uma folha. Um valor pequeno pode causar overfitting. Um valor maior simplifica a árvore, evitando folhas muito específicas e melhorando a generalização.

min_samples_split define o número mínimo de amostras necessárias para dividir um nó. Um valor pequeno melhora o detalhamento ou leva ao overfitting. Um valor maior faz com que não haja divisões desnecessárias, melhorando a generalização.

Q4-

Uma breve explicação dos otimizadores

GridSearchCV realiza uma busca exaustiva sobre valores de parâmetros especificados. Ele avalia todas as combinações possíveis de hiperparâmetros usando validação cruzada. Ele é útil quando o espaço de busca de hiperparâmetros é pequeno e pode ser explorado completamente.

RandomizedSearchCV avalia um número fixo de combinações de hiperparâmetros a partir de distribuições especificadas. É mais eficiente que GridSearchCV para grandes espaços de busca, pois não avalia todas as combinações.

BayesSearchCV usa técnicas de otimização bayesiana para encontrar os melhores hiperparâmetros. Ele constrói um modelo probabilístico da função objetivo e usa esse modelo para selecionar os hiperparâmetros mais promissores para avaliação. É eficiente para espaços de busca grandes e contínuos.

O código:

No código primeiramente avaliamos a base de dados para observarmos valores faltantes e excluímos colunas que são desnecessárias.

```
base_treino.drop(columns=['Name', 'Ticket', 'Cabin', 'Embarked', 'PassengerId'], inplace=True)
✓ 0.0s
```

Após essa exclusão a única coluna que ainda possuía valores faltantes era a coluna de idade, por causa disso decidi preencher ela com a média de todas as idades.

```
media_idade = round(base_treino['Age'].mean())
print(media_idade)
✓ 0.0s
```

```
base_treino['Age'].fillna(value=media_idade, inplace=True)
✓ 0.0s
```

A única coluna que precisava de tratamento quanto ao seus valores foi a coluna do sexo, para alterar ela foi utilizado o LabelEncoder para a transformação de string para binario.

```
base_treino['Sex'] = LabelEncoder().fit_transform(base_treino['Sex'])
0.0s
```

A codificação definiu que homens são 1 e mulheres são 0.

Foi utilizado o train_test_split para separar os valores de treino e teste e posteriormente foram salvos utilizando o numpy

```
np.savez('Titanic.npz', X_treino=X_treino, X_teste=X_teste, y_treino=y_treino, y_teste=y_teste, feature_names=X_treino.columns)
0.0s
```

Quanto aos hiperparâmetros foi utilizado os seguintes para cada modelo de otimização:

```
param_grid = {
    'criterion':  ['gini', 'entropy'],
    'max_depth':  [None, 2, 4, 6, 8, 10],
    'max_features': [None, 'sqrt', 'log2', 0.2, 0.4, 0.6, 0.8],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

param_dist = {
    'criterion':  ['gini', 'entropy'],
    'max_depth': randint(1, 50),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 10),
}


param_bayes = {
    'criterion':  ['gini', 'entropy'],
    'max_depth': Integer(1, 50),
    'min_samples_split': Integer(2, 20),
    'min_samples_leaf': Integer(1, 10),
}
```

Quanto aos resultados:

```python
grid_search = GridSearchCV( DecisionTreeClassifier(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_treino, y_treino)
print("Melhores parâmetros encontrados com GridSearchCV:")
print(grid_search.best_params_)
print("Melhor pontuação de validação cruzada: {:.2f}".format(grid_search.best_score_))
```

✓ 3.4s

```
Melhores parâmetros encontrados com GridSearchCV:
{'criterion': 'gini', 'max_depth': 10, 'max_features': 0.8, 'min_samples_leaf': 4, 'min_samples_split': 2}
Melhor pontuação de validação cruzada: 0.83
```

```python
random_search = RandomizedSearchCV(DecisionTreeClassifier(), param_dist, n_iter=100, cv=5, scoring='accuracy', random_state=42)
random_search.fit(X_treino, y_treino)
print("\nMelhores parâmetros encontrados com RandomizedSearchCV:")
print(random_search.best_params_)
print("Melhor pontuação de validação cruzada: {:.2f}".format(random_search.best_score_))
```

✓ 0.5s

```
Melhores parâmetros encontrados com RandomizedSearchCV:
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
Melhor pontuação de validação cruzada: 0.82
```

```python
bayes_search = BayesSearchCV(DecisionTreeClassifier(), param_bayes, n_iter=50, cv=5, scoring='accuracy', random_state=42)
bayes_search.fit(X_treino, y_treino)
print("\nMelhores parâmetros encontrados com BayesSearchCV:")
print(bayes_search.best_params_)
print("Melhor pontuação de validação cruzada: {:.2f}".format(bayes_search.best_score_))
```

✓ 17.8s

```
Melhores parâmetros encontrados com BayesSearchCV:
OrderedDict({'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 10, 'min_samples_split': 12})
Melhor pontuação de validação cruzada: 0.82
```