breat_cancer_ensemble_learning

May 14, 2024

1 Análise preditiva para prever o diagnóstico de câncer de mama

De Ihan Messias

1. Identificando o Problema:

   O câncer de mama é a forma mais comum de câncer em mulheres nos Estados Unidos, representando quase um terço de todos os diagnósticos. Além disso, é a segunda principal causa de morte relacionada ao câncer entre as mulheres. Essa condição surge devido ao crescimento anormal de células no tecido mamário, formando tumores que podem variar de benignos a malignos. O diagnóstico é realizado por meio de diversos exames, como mamografia, ressonância magnética, ultrassom e biópsia.

2. Objetivo: O problema se enquadra em duas categorias:

   - 1 = Maligno (canceroso) — Detectado
   - 0 = Benigno (não canceroso) — Ausente

   `No contexto do aprendizado de máquina, isso representa um problema de classificação.`

   Nosso objetivo é criar um modelo inteligente capaz de determinar se um caso de câncer de mama é benigno ou maligno, ou seja, se é inofensivo ou potencialmente perigoso, ao receber novos dados. Utilizaremos um método chamado classificação, que é essencialmente baseado em fórmulas matemáticas. Essas fórmulas aprendem com dados anteriores e, quando fornecemos novos dados, conseguem fazer previsões precisas sobre a natureza do câncer: benigno ou maligno.

   Nosso modelo representa uma ferramenta inestimável para médicos e profissionais de saúde, oferecendo insights confiáveis e fundamentados em dados. Desempenha um papel crucial na identificação precisa do câncer de mama, permitindo diagnósticos mais rápidos e precisos. Essa precisão é essencial para um tratamento eficaz, proporcionando aos pacientes uma melhor chance de recuperação.

3. Localização dos Dados:

   A fonte dos dados que utilizamos está disponível em um repositório de aprendizado de máquina mantido pela Universidade da Califórnia, Irvine. Este conjunto de dados é composto por 569 amostras de células tumorais.

   Clique aqui para baixar os dados pelo kaggle

1.1 Análise exploratória de dados

[ ]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt # v3.7.3
import seaborn as sns
```

[ ]:
```
# inserting dataframe
df = pd.read_csv('./data.csv')
```

[ ]:
```
# capture dimention
l,c = df.shape
print(f'linhas: {l} | colunas: {c}')
```

```
linhas: 569 | colunas: 33
```

[ ]:
```
# checking missing values
df.isnull().sum()
```

[ ]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
```

```
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

[ ]:
```
# remove columns
df.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)
df.columns
```

[ ]:
```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

[ ]:
```
# descriptive statistics
df.describe().T
```

[ ]:

	count	mean	std	min	\
radius_mean	569.0	14.127292	3.524049	6.981000	
texture_mean	569.0	19.289649	4.301036	9.710000	
perimeter_mean	569.0	91.969033	24.298981	43.790000	
area_mean	569.0	654.889104	351.914129	143.500000	
smoothness_mean	569.0	0.096360	0.014064	0.052630	
compactness_mean	569.0	0.104341	0.052813	0.019380	
concavity_mean	569.0	0.088799	0.079720	0.000000	
concave points_mean	569.0	0.048919	0.038803	0.000000	
symmetry_mean	569.0	0.181162	0.027414	0.106000	
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	
radius_se	569.0	0.405172	0.277313	0.111500	
texture_se	569.0	1.216853	0.551648	0.360200	
perimeter_se	569.0	2.866059	2.021855	0.757000	
area_se	569.0	40.337079	45.491006	6.802000	
smoothness_se	569.0	0.007041	0.003003	0.001713	
compactness_se	569.0	0.025478	0.017908	0.002252	
concavity_se	569.0	0.031894	0.030186	0.000000	
concave points_se	569.0	0.011796	0.006170	0.000000	
symmetry_se	569.0	0.020542	0.008266	0.007882	
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	

radius_worst	569.0	16.269190	4.833242	7.930000
texture_worst	569.0	25.677223	6.146258	12.020000
perimeter_worst	569.0	107.261213	33.602542	50.410000
area_worst	569.0	880.583128	569.356993	185.200000
smoothness_worst	569.0	0.132369	0.022832	0.071170
compactness_worst	569.0	0.254265	0.157336	0.027290
concavity_worst	569.0	0.272188	0.208624	0.000000
concave points_worst	569.0	0.114606	0.065732	0.000000
symmetry_worst	569.0	0.290076	0.061867	0.156500
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040

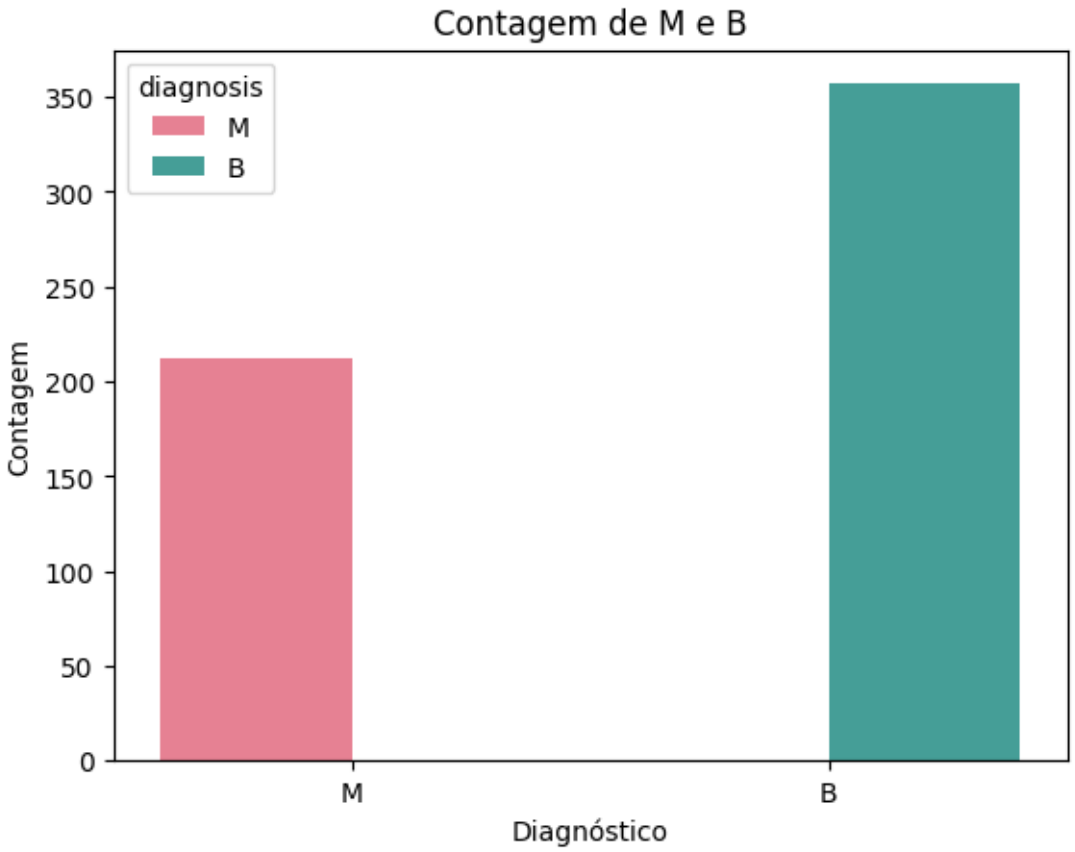
	25%	50%	75%	max
radius_mean	11.700000	13.370000	15.780000	28.11000
texture_mean	16.170000	18.840000	21.800000	39.28000
perimeter_mean	75.170000	86.240000	104.100000	188.50000
area_mean	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	0.086370	0.095870	0.105300	0.16340
compactness_mean	0.064920	0.092630	0.130400	0.34540
concavity_mean	0.029560	0.061540	0.130700	0.42680
concave points_mean	0.020310	0.033500	0.074000	0.20120
symmetry_mean	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	0.057700	0.061540	0.066120	0.09744
radius_se	0.232400	0.324200	0.478900	2.87300
texture_se	0.833900	1.108000	1.474000	4.88500
perimeter_se	1.606000	2.287000	3.357000	21.98000
area_se	17.850000	24.530000	45.190000	542.20000
smoothness_se	0.005169	0.006380	0.008146	0.03113
compactness_se	0.013080	0.020450	0.032450	0.13540
concavity_se	0.015090	0.025890	0.042050	0.39600
concave points_se	0.007638	0.010930	0.014710	0.05279
symmetry_se	0.015160	0.018730	0.023480	0.07895
fractal_dimension_se	0.002248	0.003187	0.004558	0.02984
radius_worst	13.010000	14.970000	18.790000	36.04000
texture_worst	21.080000	25.410000	29.720000	49.54000
perimeter_worst	84.110000	97.660000	125.400000	251.20000
area_worst	515.300000	686.500000	1084.000000	4254.00000
smoothness_worst	0.116600	0.131300	0.146000	0.22260
compactness_worst	0.147200	0.211900	0.339100	1.05800
concavity_worst	0.114500	0.226700	0.382900	1.25200
concave points_worst	0.064930	0.099930	0.161400	0.29100
symmetry_worst	0.250400	0.282200	0.317900	0.66380
fractal_dimension_worst	0.071460	0.080040	0.092080	0.20750

[ ]:
```
# check diagnostic values
df.diagnosis.value_counts()
```

[ ]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

[ ]:
```
# diagnostic values
ax = sns.countplot(data=df, x='diagnosis', hue='diagnosis', palette='husl' )
ax.set(xlabel='Diagnóstico', ylabel='Contagem')
plt.title('Contagem de M e B')
```

[ ]:
```
Text(0.5, 1.0, 'Contagem de M e B')
```



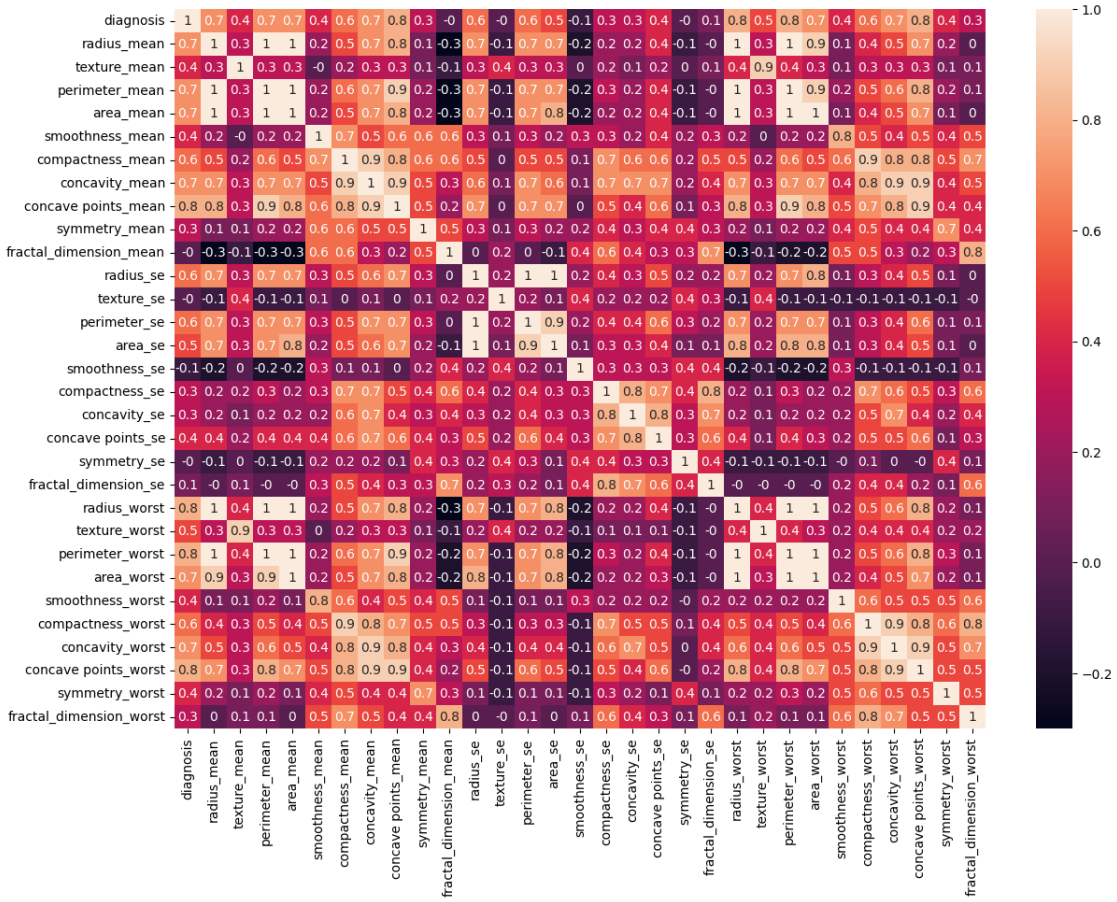
Classificação binária é mais simples de entender e implementar em comparação com problemas de classificação multiclasse.

[ ]:
```
# binary classification
df['diagnosis'] = df['diagnosis'].apply(lambda x: 1 if x == 'M' else 0)
```

[ ]:
```
# correlation
plt.figure(figsize=(14,10))
```

```
sns.heatmap(df.corr().round(1), annot=True)
```

[ ]: <Axes: >



1.2 Pré-processamento dos dados

- sklearn.model_selection.train_test_split: > Usada para dividir um conjunto de dados em conjuntos de treinamento e teste para treinar e avaliar modelos de machine learning.
- sklearn.tree.DecisionTreeClassifier: > É um algoritmo de aprendizado de máquina baseado em árvore de decisão. O modelo de DecisionTreeClassifier cria uma única árvore de decisão para fazer previsões. Ele divide o conjunto de dados em subconjuntos com base em critérios como entropia ou índice Gini, e faz previsões com base na maioria dos votos em cada folha da árvore.
- sklearn.linear_model.LogisticRegression: > É um algoritmo de aprendizado de máquina para classificação. O modelo de LogisticRegression usa uma função logística para modelar a probabilidade de uma classe específica. É comumente usado para problemas de classificação binária, mas também pode ser usado para problemas de classificação multiclasse através de técnicas como “one-vs-rest” ou “multinomial”.

- sklearn.svm.SVC: > É um algoritmo de aprendizado de máquina baseado em vetores de suporte. O modelo SVC (Support Vector Classifier) tenta encontrar o hiperplano que melhor divide as classes no espaço de características. Ele usa técnicas como "kernel trick" para lidar com dados não linearmente separáveis.
- sklearn.ensemble.RandomForestClassifier: > Algoritmo de aprendizado de máquina baseado em árvores de decisão. O modelo de RandomForest cria várias árvores de decisão e as combina para obter previsões mais precisas (Usa a média das previsões das árvores individuais para fazer previsões finais).
- sklearn.ensemble.VotingClassifier: > É um algoritmo de aprendizado de máquina que combina previsões de múltiplos modelos de aprendizado de máquina. O modelo de VotingClassifier faz previsões com base na maioria dos votos dos modelos individuais. Ele pode usar votação "hard", onde a classe com a maioria dos votos é escolhida, ou votação "soft", onde as probabilidades de classe são somadas e a classe com a maior soma é escolhida. É útil para aumentar a robustez e a precisão do modelo ao combinar as forças de vários modelos diferentes.
- sklearn.metrics.classification_report: > Relatório de métricas de classificação, incluindo precisão, recall e F1-score para avaliar o desempenho de um modelo de classificação.
- sklearn.metrics.confusion_matrix: > Matriz de confusão que mostra as contagens de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos.

[ ]:
```
# import machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier # Model 1
from sklearn.linear_model import LogisticRegression # Model 2
from sklearn.svm import SVC # Model 3
from sklearn.ensemble import RandomForestClassifier # Model 4
from sklearn.ensemble import VotingClassifier

from sklearn.metrics import classification_report, confusion_matrix
```

[ ]:
```
# dataset split
X = df.drop(columns=['diagnosis']) # dataset independent variables (M,B)
y = df['diagnosis']  # Target data (what we want to predict)
# training and testing set (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

A escolha do número 42 como valor para random_state ou qualquer outro número inteiro não tem um significado especial do ponto de vista da funcionalidade do algoritmo RandomForestClassifier. O importante é que ele seja fixo para garantir que os resultados do modelo sejam reproduzíveis em diferentes execuções do código. Em programação e na comunidade de desenvolvedores, o número 42 é algumas vezes usado como uma "Resposta para a Vida, o Universo e Tudo" em uma piada popular do livro "O Guia do Mochileiro das Galáxias" de Douglas Adams. No contexto da pergunta "qual é a resposta para a vida, o universo e tudo mais?", a resposta foi calculada por um supercomputador como sendo o número 42, embora o significado exato disso seja um mistério, conforme a narrativa do livro.

1.3 Construção dos Modelos

A razão para usar essa metodologia é que ela pode melhorar a precisão do modelo. Cada modelo individual pode ser bom em capturar certos padrões nos dados, mas pode perder outros. Ao combinar vários modelos, você pode aproveitar os pontos fortes de cada modelo para obter um modelo final que é mais preciso e robusto. Além disso, isso pode ajudar a evitar o sobreajuste, pois os erros de um modelo podem ser compensados pelos acertos de outro modelo. Isso é conhecido como ensemble learning.

[ ]:
```
tree_clf = DecisionTreeClassifier()
lr_clf = LogisticRegression(max_iter=3000)
svm_clf = SVC()
rf_clf = RandomForestClassifier(random_state=42)

voting_clfs = VotingClassifier(estimators=[('tree', tree_clf), ('lr', lr_clf),␣
 ↪('svm', svm_clf), ('rf', rf_clf)], voting='hard')
voting_clfs.fit(X_train, y_train)
```

[ ]:
```
VotingClassifier(estimators=[('tree', DecisionTreeClassifier()),
                             ('lr', LogisticRegression(max_iter=3000)),
                             ('svm', SVC()),
                             ('rf', RandomForestClassifier(random_state=42))])
```

O parâmetro max_iter=3000 define o número máximo de iterações para o solucionador convergir. Se o modelo não convergir em 3000 iterações, ele irá parar e retornar um aviso. Aumentar este valor pode permitir que o modelo convirja para uma solução mais precisa, mas também pode aumentar o tempo de treinamento.

Foi usado para corrigir o erro: "lbfgs failed to converge (status=1): STOP: TOTAL NO. of ITERATIONS REACHED LIMIT."

O erro ocorre quando o algoritmo de otimização L-BFGS, usado na regressão logística, não consegue encontrar a solução ótima dentro do número máximo de iterações permitidas. Por padrão, o número máximo de iterações é definido como um valor relativamente baixo (por exemplo, 100). Se o algoritmo não convergir para a solução ótima dentro desse limite, ele irá parar e retornar a mensagem de erro. Aumentar o número máximo de iterações usando o parâmetro max_iter=3000 dá ao algoritmo mais tempo para encontrar a solução mais adequada.

1.4 Avaliação dos Modelos

Acurácia: Ela representa a proporção de previsões corretas feitas pelo modelo em relação ao total de previsões. Em outras palavras, a acurácia mede a precisão global do modelo.

A fórmula para calcular a acurácia é:

$$Acurácia = \frac{Previsões\ Corretas}{Total\ de\ Previsões}$$

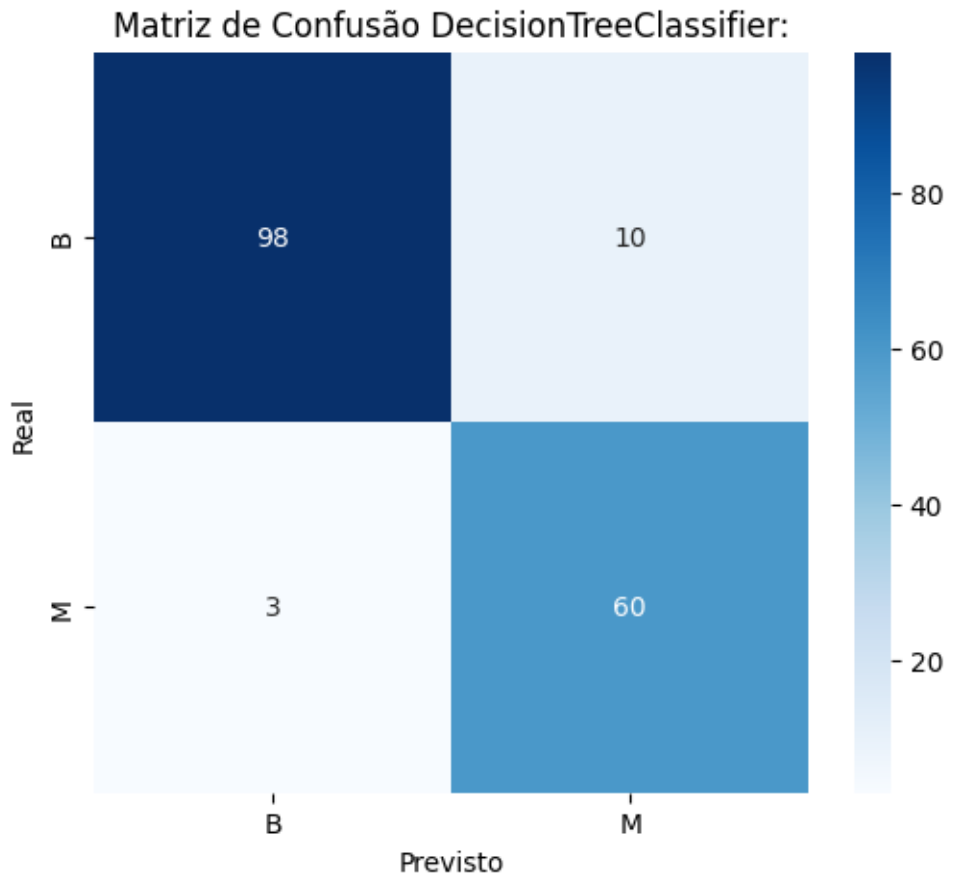
Classificação:

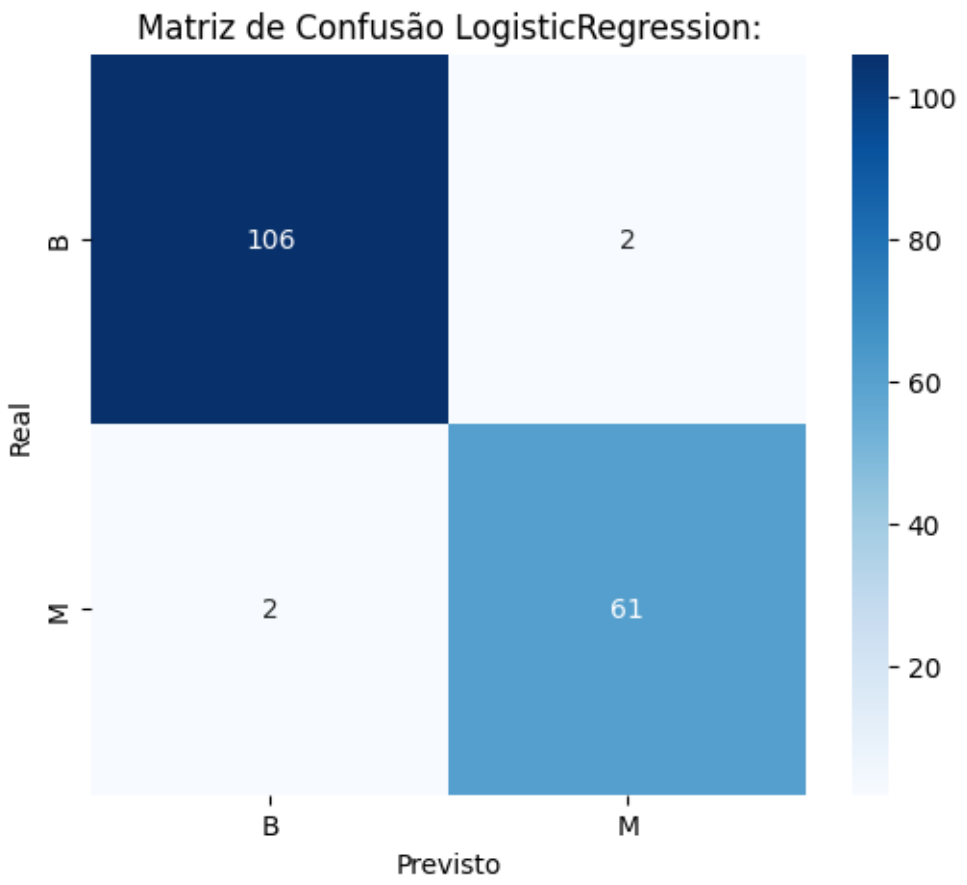
- Precision (Precisão): É a porcentagem de previsões positivas corretas em relação ao total de previsões positivas feitas pelo modelo. Quanto mais alta, menos falsos positivos o modelo tem.
- Recall (Sensibilidade): É a porcentagem de instâncias positivas reais que o modelo conseguiu capturar corretamente. Quanto mais alta, menos falsos negativos o modelo tem.
- F1-Score: É uma média equilibrada entre precisão e recall. É útil quando você quer uma métrica única que leve em consideração tanto falsos positivos quanto falsos negativos.

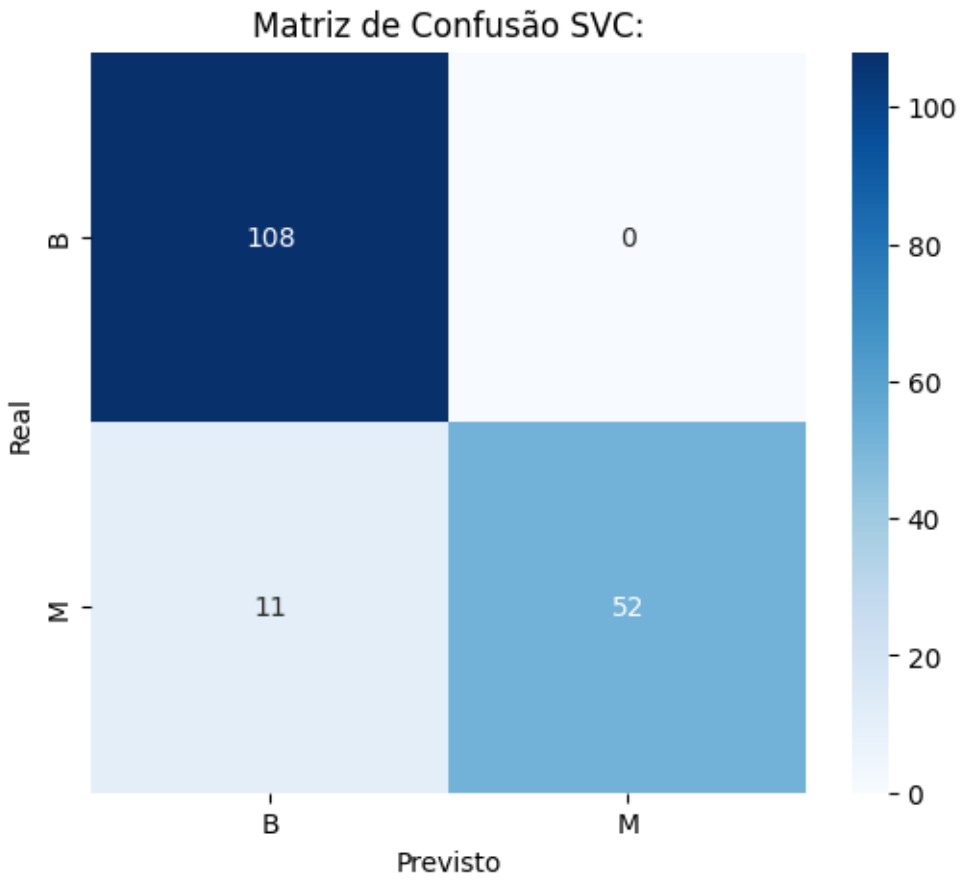
[ ]:
```
# classification_report and confusion matrix
for clf in (tree_clf, lr_clf, svm_clf, rf_clf, voting_clfs):
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    print(f'{clf.__class__.__name__}\n',classification_report(y_test,␣
 ↪predictions))

    plt.figure(figsize=(6, 5))
    sns.heatmap(confusion_matrix(y_test, predictions),
        annot=True, fmt='d', cmap='Blues',
        xticklabels=['B', 'M'], yticklabels=['B', 'M'])
    plt.xlabel('Previsto')
    plt.ylabel('Real')
    plt.title(f'Matriz de Confusão {clf.__class__.__name__}:')
```

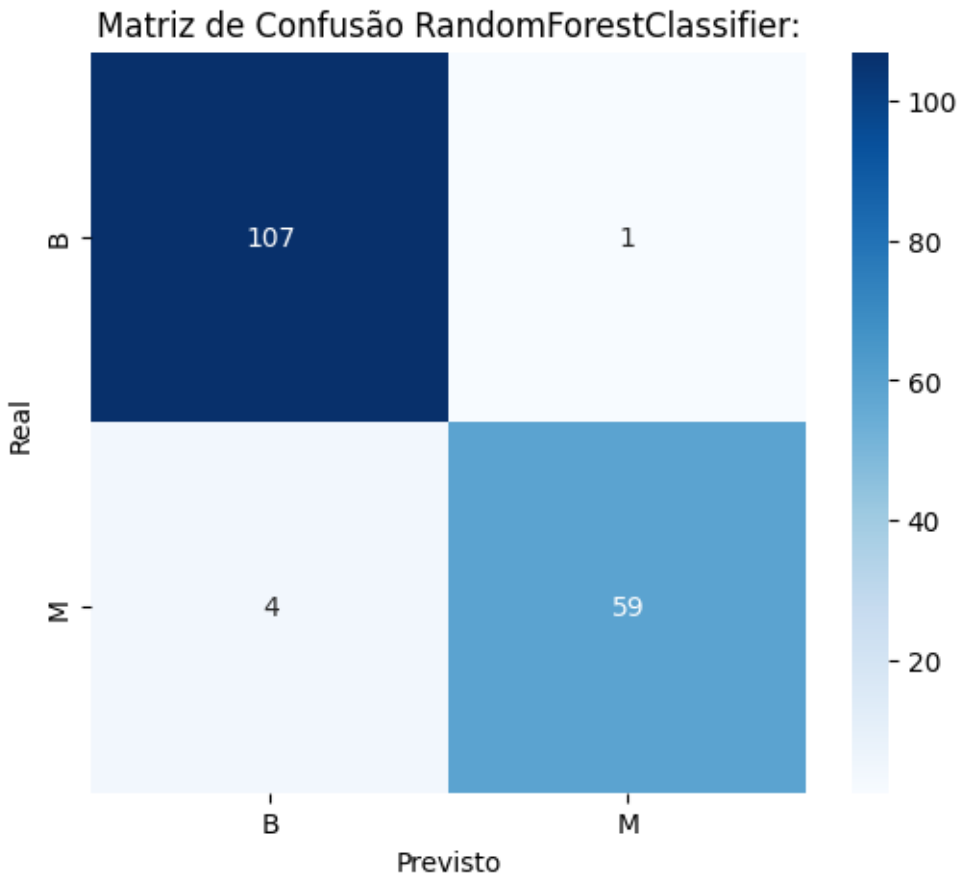
```
DecisionTreeClassifier
              precision    recall  f1-score   support

           0       0.97      0.91      0.94       108
           1       0.86      0.95      0.90        63

    accuracy                           0.92       171
   macro avg       0.91      0.93      0.92       171
weighted avg       0.93      0.92      0.92       171

LogisticRegression
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       108
           1       0.97      0.97      0.97        63

    accuracy                           0.98       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171

SVC
              precision    recall  f1-score   support
```

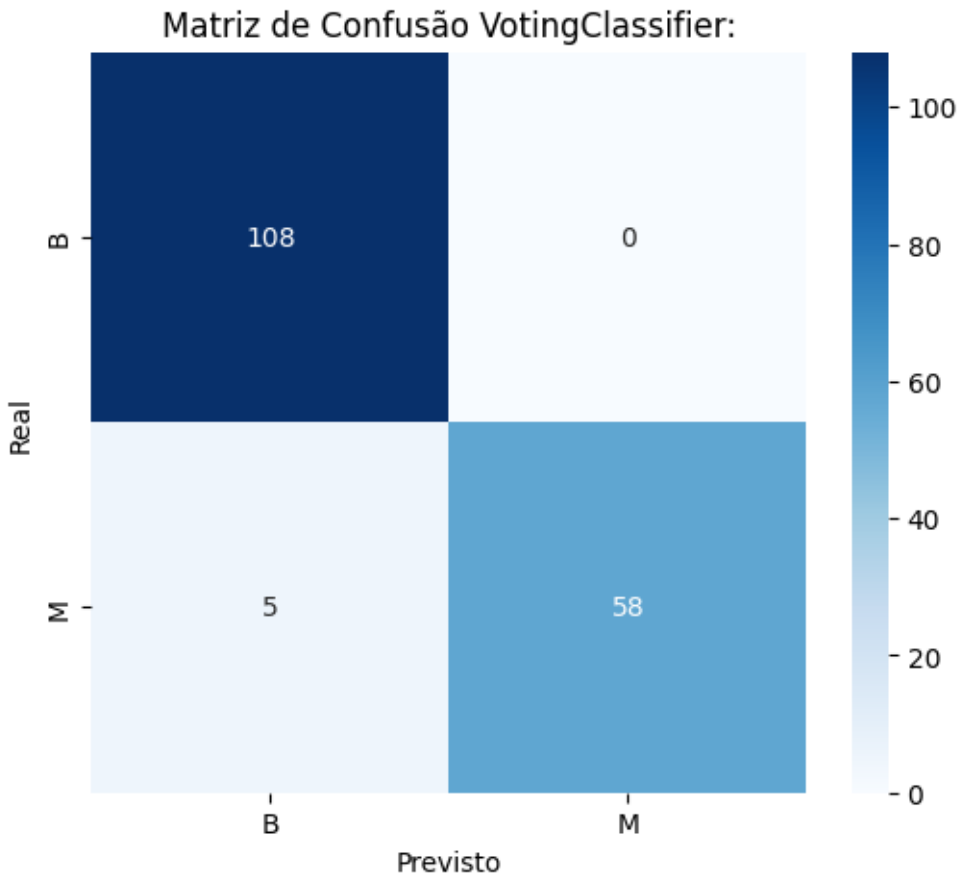
```
              0       0.91      1.00      0.95       108
              1       1.00      0.83      0.90        63

       accuracy                           0.94       171
      macro avg       0.95      0.91      0.93       171
   weighted avg       0.94      0.94      0.93       171

RandomForestClassifier
                  precision    recall  f1-score   support

              0       0.96      0.99      0.98       108
              1       0.98      0.94      0.96        63

       accuracy                           0.97       171
      macro avg       0.97      0.96      0.97       171
   weighted avg       0.97      0.97      0.97       171

VotingClassifier
                  precision    recall  f1-score   support

              0       0.96      1.00      0.98       108
              1       1.00      0.92      0.96        63

       accuracy                           0.97       171
      macro avg       0.98      0.96      0.97       171
   weighted avg       0.97      0.97      0.97       171
```


Matriz de Confusão DecisionTreeClassifier:
98
10
3
60
B
M
Real
B
M
Previsto
80
60
40
20


Matriz de Confusão LogisticRegression:
106
2
2
61
B
M
Real
B
M
Previsto
100
80
60
40
20


Matriz de Confusão SVC:
108
0
11
52
B
M
Real
B
M
Previsto
100
80
60
40
20
0


Matriz de Confusão RandomForestClassifier:
107
1
4
59
B
M
Real
B
M
Previsto
100
80
60
40
20



- Verdadeiros positivos (**B**, **B**): O modelo previu corretamente "n" instâncias como pertencentes à classe B.
- Falsos positivos (**B**, **M**): O modelo previu erroneamente "n" instâncias como pertencentes à classe B, quando na verdade pertencem à classe M.
- Falsos negativos (**M**, **B**): O modelo previu erroneamente "n" instâncias como pertencentes à classe M, quando na verdade pertencem à classe B.
- Verdadeiros negativos (**M**, **M**): O modelo previu corretamente "n" instâncias como pertencentes à classe M.