Trabalho A2 - ML

Carlos César de O. Fonseca

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
        from funcoes import evalu
```

Contextualizando

O meu trabalho é sobre Dados que foram coletados apartir de uma imagem digitalizada de um aspirado por agulha fina (PAAF) de uma massa mamária. Eles descrevem características dos núcleos celulares presentes na imagem.

Apartir dai, temos vários dados quantitativos que representam algumas coisas e temos uma coluna de dados qualitativa, os dados do Diagnostico. Esses dados a principio era uma coluna de B's e M's, onde B significava Benigno, enquanto M significava Maligno. Porém para rodar os modelos e desenvolver uma forma de avaliar se o tumor é benigno ou maligno dado apenas os dados quantitativos da interpretação da imagem.

O objetivo ao fazer a analise dos dados era prever prever os malignos como malignos e os benignos como benignos ou seja a métrica de interesse era a acurácia.

Antes de falar de acurácia, vamos definir algumas coisas:

- VP (Verdadeiro positivo) será quando o modelo prever M e target for M
- FP (Falso Positivo) será quando o modelo prever M porém o target é B
- FN (Falso Negativo) será quando o modelo prever B enquanto o real é M
- VN (Verdadeiro Negativo) será quando o modelo prever B e realmente for B

Portanto a acurácia seria:

$$Acuracia = \frac{VP+VN}{VP+FP+FN+VN}$$

Esse valor é legal de ser analisado e também diz bastante sobre a quantidade de acertos que o modelo tem. Porém, acho que seria bom analisar outras métricas como Precisão e Recall.

$$PrecisãoM = \frac{VP}{VP+FP}$$

$$RecallM = \frac{VP}{VP+FN}$$

Acho que dados os meus dados é tão importante observar o Recall de Malignos, quanto analisar a Acurácia. Os meus dados tem 357 benignos enquanto tem 212 malignos como target, acho que não é um problema para predizer, porém é necessário ter cuidado na hora de analisar os resultados, pois os dados cerca de $\frac{3}{5}$ são Benignos enquanto apenas cerca $\frac{2}{5}$ são malignos, tem um desbalanciamento, porém não é tão grande e acredito que apenas tomando cuidado com as medidas de avaliação seja possível desenvolver o melhor modelo. Além disso, acho importante analisar o Recall dos Malignos, pois a previsão de Falsos Negativos é mais negativo que os Falsos Positivos, FP acarretam alguns danos como uma ansiedade, preocupação das pessoas que foram diagnosticadas e da família dela, porém o FN pode ser pior, a pessoa em questão acredita que o tumor é benigno e para o tratamento, levando em um aumento da doença e numa menor chance de cura, possivelmente levando a óbito.

Resumo dos Passos...

- Ler os dados
    - Trocar M por 1 e B por 0 na variável diagnostico
- Separar os dados entre treino e teste
- Testar alguns modelos com os dados
- Escolher o melhor modelo e chegar a uma conclusão

Leitura dos Dados

Aqui abrimos os dados que já utilizamos no trabalho 1 e convertemos B para 0 e M para 1:

```
In [2]: df2 = pd.read_csv('cancer.csv')
        df2['diagnosis'] = df2['diagnosis'].replace('M', 1)
        df2['diagnosis'] = df2['diagnosis'].replace('B', 0)
        df2.diagnosis.astype('int64')
        df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   id                      569 non-null    int64  
 1   diagnosis               569 non-null    int64  
 2   radius_mean             569 non-null    float64
 3   texture_mean            569 non-null    float64
 4   perimeter_mean          569 non-null    float64
 5   area_mean               569 non-null    float64
 6   smoothness_mean         569 non-null    float64
 7   compactness_mean        569 non-null    float64
 8   concavity_mean          569 non-null    float64
 9   concave points_mean     569 non-null    float64
 10  symmetry_mean           569 non-null    float64
 11  fractal_dimension_mean  569 non-null    float64
dtypes: float64(10), int64(2)
memory usage: 53.5 KB
```

Observando os valores que usaremos para prever, ou seja, o X:

```
In [3]: df2.iloc[:,2:]
```

Out[3]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	
...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	

569 rows × 10 columns

Os dados qualitativos do diagnostico, ou seja, o y:

```
In [4]: df2.iloc[:,1]
```

```
Out[4]: 0      1
        1      1
        2      1
        3      1
        4      1
              ..
        564    1
        565    1
        566    1
        567    1
        568    0
        Name: diagnosis, Length: 569, dtype: int64
```

Treino e Teste

Pegando o X e o y e separando 70 dos dados para treinar os modelos, enquanto os outros 30 estão destinados para testar o modelo e nós dizer uma métrica.

```
In [5]: from sklearn import model_selection

        X_train, X_test, y_train, y_test = model_selection.train_test_split(df2.iloc[:,2:],
                                                                             df2.iloc[:,1],
                                                                             test_size=0.3,
                                                                             random_state=0)
```

```
In [6]: resultados = pd.DataFrame(columns=(['modelo','acuracia','recall']))
```

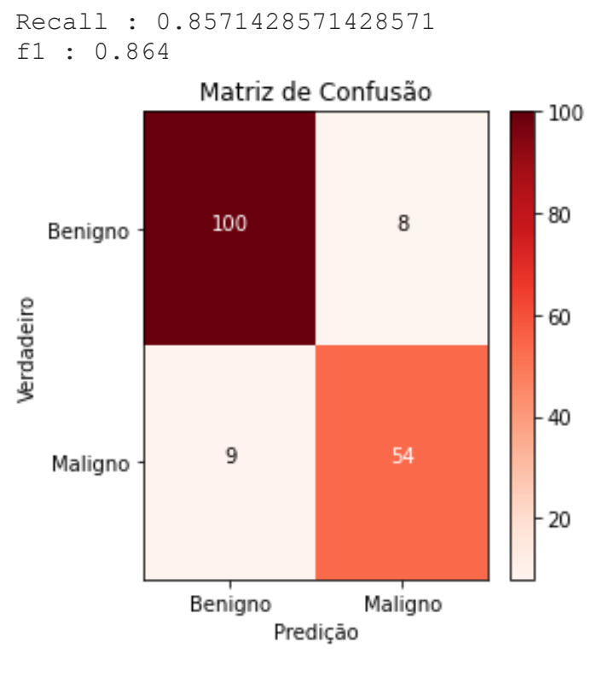
Testar Modelos

Para efeito de organização e tempo de execução para organizar este arquivo vou deixar o Grid Search apenas para o modelo KNN, porém foi usado para encontrar os melhores parametros para todos os modelos.

Naive Bayes

```
In [7]: from sklearn.naive_bayes import GaussianNB

        gnb = GaussianNB().fit(X_train, y_train)

        resultados = evalu(gnb, X_test, y_test,'Naive Bayes', resultados)
```

```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.92      0.93      0.92       108
     Maligno       0.87      0.86      0.86        63

    accuracy                           0.90       171
   macro avg       0.89      0.89      0.89       171
weighted avg       0.90      0.90      0.90       171

ROC: 0.8915343915343915
Acurácia : 0.9005847953216374
Precisão : 0.7991753577492118
Recall : 0.8571428571428571
f1 : 0.864
```



KNN

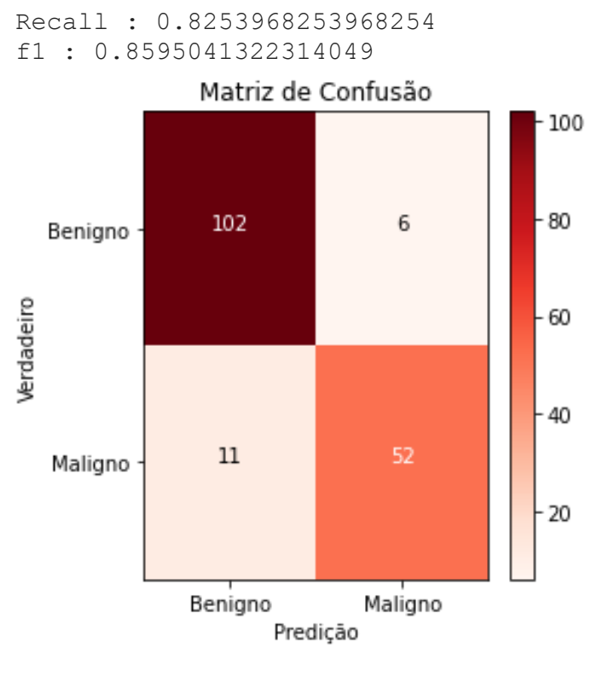
```
In [8]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [9]: E = KNeighborsClassifier()
        P = {'n_neighbors':[5,10,15,20,25,35,45,50,55,65,75],'weights':['uniform','distance']
        k = model_selection.GridSearchCV(estimator=E,
                                         param_grid=P,
                                         scoring='recall',
                                         n_jobs=-1).fit(X_train, y_train)
        k.best_estimator_
```

```
Out[9]: KNeighborsClassifier(weights='distance')
```

```
In [10]: knn = KNeighborsClassifier(weights='distance').fit(X_train, y_train)

         resultados = evalu(knn, X_test, y_test,'KNN',resultados)
```

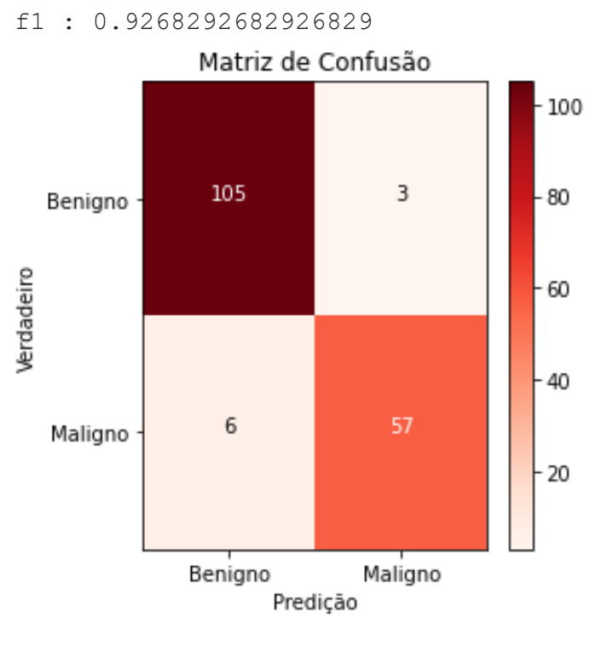
```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.90      0.94      0.92       108
     Maligno       0.90      0.83      0.86        63

    accuracy                           0.90       171
   macro avg       0.90      0.88      0.89       171
weighted avg       0.90      0.90      0.90       171

ROC: 0.8849206349206349
Acurácia : 0.9005847953216374
Precisão : 0.8043384322876156
Recall : 0.8253968253968254
f1 : 0.8595041322314049
```



Logistic Regression

```
In [11]: from sklearn.linear_model import LogisticRegression

         LR = LogisticRegression(C=7.702999999999999, multi_class='ovr', penalty='l1',
                      solver='liblinear', tol=0.0033333334000000005).fit(X_train, y_trai

         resultados= evalu(LR, X_test, y_test,'Reg. Logistica',resultados)
```

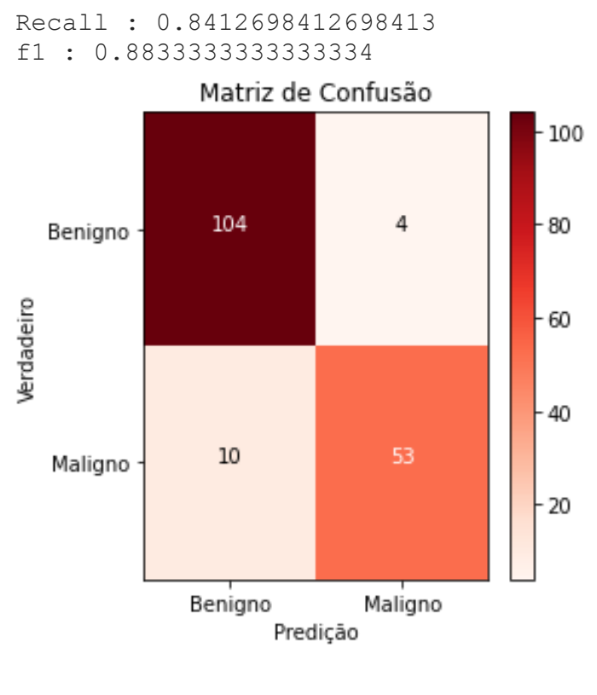
```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.95      0.97      0.96       108
     Maligno       0.95      0.90      0.93        63

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171

ROC: 0.9384920634920634
Acurácia : 0.9473684210526315
Precisão : 0.8946115288220551
Recall : 0.9047619047619048
f1 : 0.9268292682926829
```



Arvore de Decisão

```
In [12]: from sklearn.tree import DecisionTreeClassifier
         dtc = DecisionTreeClassifier(random_state=42,
                                      splitter='random'
                                      ).fit(X_train, y_train)

         resultados = evalu(dtc, X_test, y_test,'Arvore de Decisão',resultados)
```

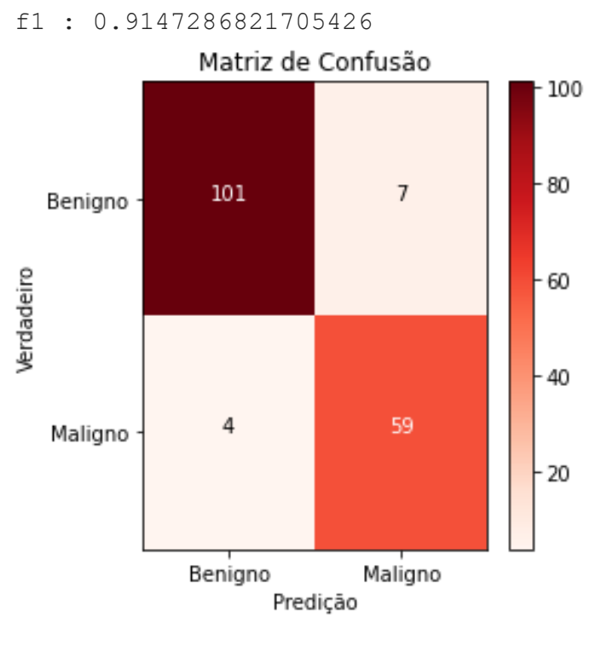
```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.91      0.96      0.94       108
     Maligno       0.93      0.84      0.88        63

    accuracy                           0.92       171
   macro avg       0.92      0.90      0.91       171
weighted avg       0.92      0.92      0.92       171

ROC: 0.9021164021164022
Acurácia : 0.9181286549707602
Precisão : 0.8407128933444723
Recall : 0.8412698412698413
f1 : 0.8833333333333334
```



Random Forest

```
In [13]: from sklearn.ensemble import RandomForestClassifier
         rf = RandomForestClassifier(criterion='entropy', max_depth=6, n_estimators=200,
                                     random_state=42).fit(X_train, y_train)


         resultados = evalu(rf, X_test, y_test, 'Random Forest', resultados)
```

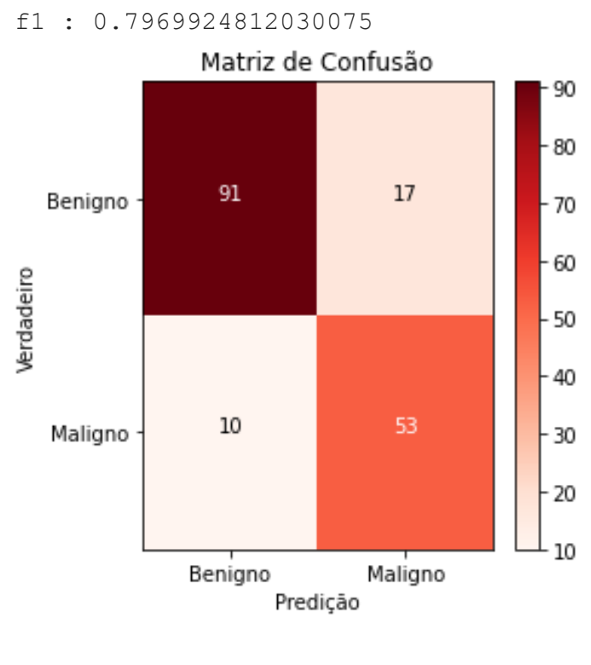
```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.96      0.94      0.95       108
     Maligno       0.89      0.94      0.91        63

    accuracy                           0.94       171
   macro avg       0.93      0.94      0.93       171
weighted avg       0.94      0.94      0.94       171

ROC: 0.9358465608465609
Acurácia : 0.935672514619883
Precisão : 0.8605731500468342
Recall : 0.9365079365079365
f1 : 0.9147286821705426
```



SGDC Classifier

```
In [14]: from sklearn.linear_model import SGDClassifier
         sgdc = SGDClassifier(alpha=1, loss='squared_hinge', max_iter=20, penalty='l1').fit(X_t

         resultados = evalu(sgdc, X_test, y_test,'SGDC',resultados)
```

```
C:\Users\carlo\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_gradient.p
y:574: ConvergenceWarning: Maximum number of iteration reached before convergence. Con
sider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
Resultados:
               precision    recall  f1-score   support

     Benigno       0.90      0.84      0.87       108
     Maligno       0.76      0.84      0.80        63

    accuracy                           0.84       171
   macro avg       0.83      0.84      0.83       171
weighted avg       0.85      0.84      0.84       171

ROC: 0.8419312169312169
Acurácia : 0.8421052631578947
Precisão : 0.6954409834109082
Recall : 0.8412698412698413
f1 : 0.7969924812030075
```



Testando algo diferente...

Aqui, vamos testar algumas ideias e utilizar os modelos de Naive Bayes (porque é simples) e Random Forest (porque foi a que teve o melhor desempenho associando acurácia e recall) para ver se elas são mais eficientes que os modelos que temos até agora.

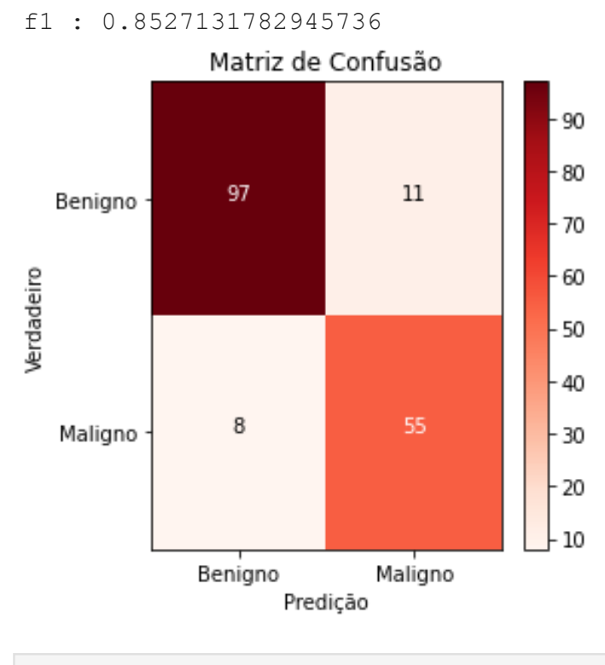
Ideia 1

Raio, Area e Perimetro são altamente correlacionados, como eu disse no primeiro trabalho, acho que Area e Perimetro são derivados do Raio, então vou testar alguns modelos sem essas variavéis.

```
In [15]: df3 = df2.drop(['perimeter_mean','area_mean'],axis=1)
         X_train1, X_test1, y_train1, y_test1 = model_selection.train_test_split(df3.iloc[:,2:]
                                                                                 df3.iloc[:,1]
                                                                                 test_size=0.3
                                                                                 random_state=
```

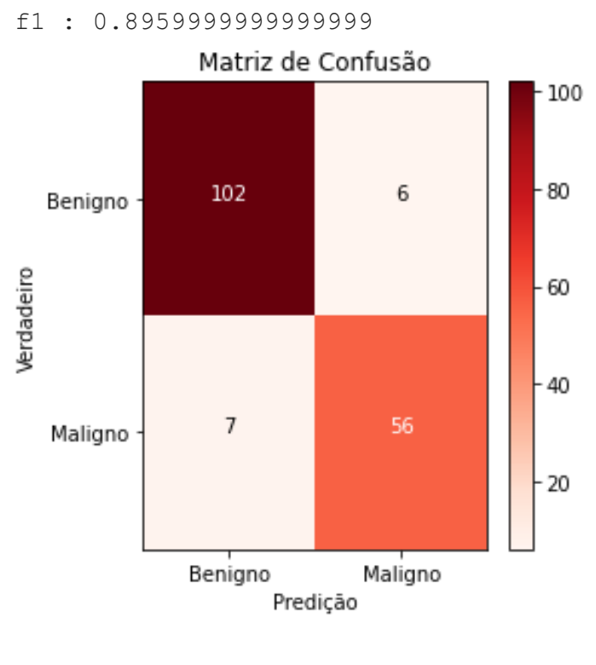
```
In [16]: gnb1 = GaussianNB().fit(X_train1, y_train1)

         resultados = evalu(gnb1, X_test1, y_test1,'NB - 1', resultados)
```

```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.92      0.90      0.91       108
     Maligno       0.83      0.87      0.85        63

    accuracy                           0.89       171
   macro avg       0.88      0.89      0.88       171
weighted avg       0.89      0.89      0.89       171

ROC: 0.8855820105820106
Acurácia : 0.8888888888888888
Precisão : 0.7742968532442217
Recall : 0.873015873015873
f1 : 0.8527131782945736
```



```
In [17]: rf1 = RandomForestClassifier(max_depth=5, n_estimators=500, random_state=42).fit(X_tra

         resultados = evalu(rf1, X_test1, y_test1,'RF - 1',resultados)
```

```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.94      0.94      0.94       108
     Maligno       0.90      0.89      0.90        63

    accuracy                           0.92       171
   macro avg       0.92      0.92      0.92       171
weighted avg       0.92      0.92      0.92       171

ROC: 0.9166666666666666
Acurácia : 0.9239766081871345
Precisão : 0.8438030560271647
Recall : 0.8888888888888888
f1 : 0.8959999999999999
```



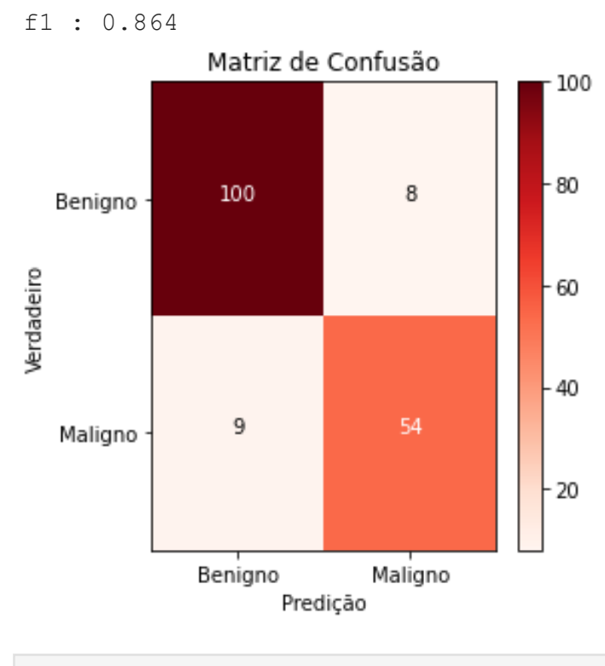
Ideia 2

E se em vez de tirar os correlacionados entre si, retirar os que tem menor correlação com a resposta. Nessa hipotese, eu retirei as variaveis que tinha uma correlação menor que 0.4 com a variavel a ser predita.

```
In [18]: df4 = df2.drop(['fractal_dimension_mean','symmetry_mean','smoothness_mean'],axis=1)
         X_train2, X_test2, y_train2, y_test2 = model_selection.train_test_split(df4.iloc[:,2:]
                                                                                 test_size=0.3,
```

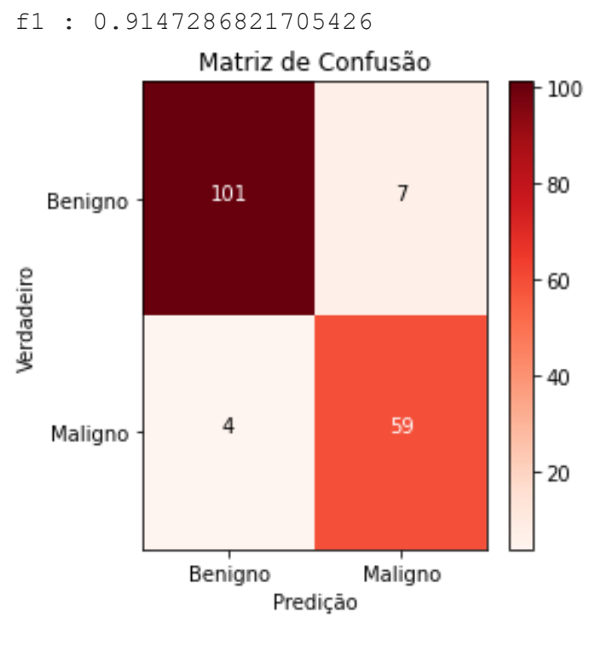
```
In [19]: gnb2 = GaussianNB().fit(X_train2, y_train2)

         resultados = evalu(gnb2, X_test2, y_test2,'NB - 2', resultados)
```

```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.92      0.93      0.92       108
     Maligno       0.87      0.86      0.86        63

    accuracy                           0.90       171
   macro avg       0.89      0.89      0.89       171
weighted avg       0.90      0.90      0.90       171

ROC: 0.8915343915343915
Acurácia : 0.9005847953216374
Precisão : 0.7991753577492118
Recall : 0.8571428571428571
f1 : 0.864
```



```
In [20]: rf2 = RandomForestClassifier(max_depth=4, n_estimators=500, random_state=42).fit(X_tra

         resultados = evalu(rf2, X_test2, y_test2,'RF - 2',resultados)
```

```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.96      0.94      0.95       108
     Maligno       0.89      0.94      0.91        63

    accuracy                           0.94       171
   macro avg       0.93      0.94      0.93       171
weighted avg       0.94      0.94      0.94       171

ROC: 0.9358465608465609
Acurácia : 0.935672514619883
Precisão : 0.8605731500468342
Recall : 0.9365079365079365
f1 : 0.9147286821705426
```



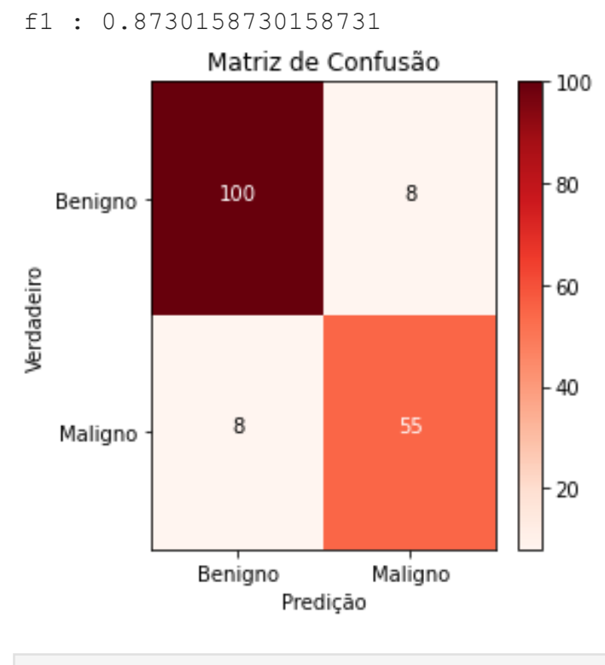
Ideia 3

E se juntar a Ideia 1 com a Ideia 2

```
In [21]: df5 = df2.drop(['perimeter_mean','area_mean','fractal_dimension_mean','symmetry_mean',
         X_train3, X_test3, y_train3, y_test3 = model_selection.train_test_split(df5.iloc[:,2:]
                                                                                 test_size=0.3,
```

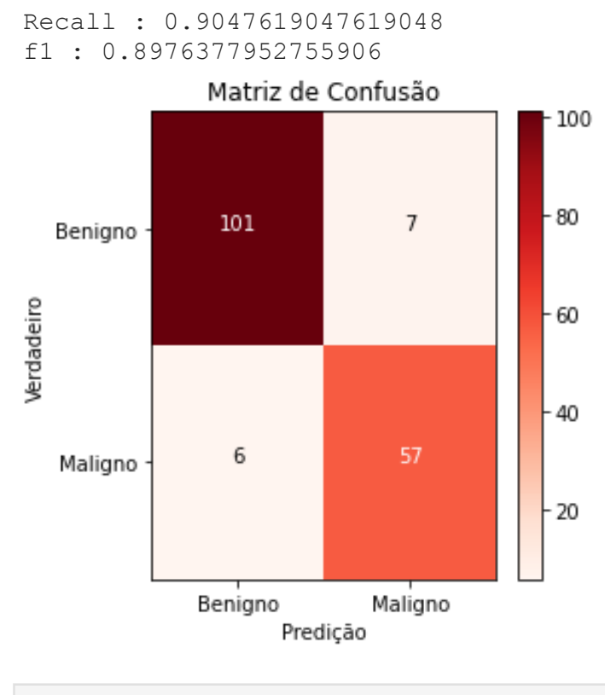
```
In [22]: gnb3 = GaussianNB().fit(X_train3, y_train3)

         resultados = evalu(gnb3, X_test3, y_test3,'NB - 3', resultados)
```

```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.93      0.93      0.93       108
     Maligno       0.87      0.87      0.87        63

    accuracy                           0.91       171
   macro avg       0.90      0.90      0.90       171
weighted avg       0.91      0.91      0.91       171

ROC: 0.8994708994708994
Acurácia : 0.9064327485380117
Precisão : 0.8089403402686611
Recall : 0.873015873015873
f1 : 0.873015873015873
```



```
In [23]: rf3 = RandomForestClassifier(max_depth=4, n_estimators=500, random_state=42).fit(X_tra

         resultados = evalu(rf3, X_test3, y_test3,'RF - 3',resultados)
```

```
Resultados:
               precision    recall  f1-score   support

     Benigno       0.94      0.94      0.94       108
     Maligno       0.89      0.90      0.90        63

    accuracy                           0.92       171
   macro avg       0.92      0.92      0.92       171
weighted avg       0.92      0.92      0.92       171

ROC: 0.919973544973545
Acurácia : 0.9239766081871345
Precisão : 0.840891290726817
Recall : 0.9047619047619048
f1 : 0.8976377952755906
```



```
In [24]: resultados
```

Out[24]:

	modelo	acuracia	recall
0	Naive Bayes	0.900585	0.857143
1	KNN	0.900585	0.825397
2	Reg. Logistica	0.947368	0.904762
3	Arvore de Decisão	0.918129	0.841270
4	Random Forest	0.935673	0.936508
5	SGDC	0.842105	0.841270
6	NB - 1	0.888889	0.873016
7	RF - 1	0.923977	0.888889
8	NB - 2	0.900585	0.857143
9	RF - 2	0.935673	0.936508
10	NB - 3	0.906433	0.873016
11	RF - 3	0.923977	0.904762

Conclusão

Acredito que os modelos foram satisfatórios, e eu diria que a Random Forest, foi o modelo que melhor previu os dados. Isso porque mesmo não tendo a melhor acurácia, teve o melhor recall e teve uma acurácia muito próxima do modelo que teve a maior.

Além disso, eu queria resaltar um importante ponto, o modelo deu certo tem uma taxa de acerto bem grande ao predizer se o tumor é maligno ou benigno, porém ele é inútil para pessoas "normais", para ele funcionar relativamente bem, as pessoas que vão sofrer a avaliação do modelo tem estar minimamente padronizadas com os dados que foram dados para modelar a questão. Como os dados são de exames de cancer de mama com pessoas que já tem o cancer diagnosticado, não é conveniente chegar e em pessoas que não estão sentido nada e fazer um teste com elas, o modelo com certeza perderia a eficiencia, pois os pre reqisitos na hora de colher os dados não estaram sendo compridos.

Outros:

Estão no GITHUB:

- O arquivo funcoes contem as funções que foram usadas para avaliar o modelo.
- O notebook com todos os passos, (aqui eu resumi um pouco de código para não poluir tanto)

```
In [25]: #FIM
```